

SDK-86

SYSTEM DESIGN KERNEL

USER'S MANUAL

Additional copies of this manual may be purchased from:

URDA, Inc.
4516 Henry Street, Suite 407
Pittsburgh, PA 15213

URDA retains the right to change the specifications at any time, without notice.

This product is manufactured by URDA, Inc. under license from Intel Corporation

Neither URDA or Intel makes any warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Neither URDA, Inc. or Intel Corporation assumes any responsibility for any errors that may appear in this document. Neither URDA, Inc. or Intel Corporation makes any commitment to update or to keep current the information contained in this document.

Neither URDA, Inc. or Intel Corporation assumes any responsibility for the use of any circuitry other than circuitry embodied in this product. No other patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following is a trademark of Intel Corporation and its affiliates and may only be used to identify products: MCS.



PREFACE

This is the User's Manual for the SDK-86 System Design Kernel. Included within this manual are descriptions of the two ROM-resident monitor programs that allow you to communicate with your kit, a functional description of the SDK-86 itself, interfacing information and design examples for expanding your kit's capabilities. Additional information is available in the following Intel publications:

- *SDK-86 MCS-86 System Design Kit Monitor Listings, Order No. 9800699
 - *The 8086 Family User's Manual, Order No. 980072
 - *Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems, Application Note AP-30
 - ASM86 Language Reference Manual, Order No. 9800640
 - MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users, Order No. 9800639
 - 80/85 Absolute Object File Formats Technical Specification, Order No. 9800183
 - Intel Component Data Catalog
- *Supplied with SDK-86



SERVICE ASSISTANCE

STARTING THE FIRST TIME

Energize the +5 Volt power Supply.

Press the RESET button on the keyboard. The display should respond by reading out "86 ./".

If the above readout appears, go on to Chapter 3 of this book and try out each button and function. Verify that each command produces the specified result, and that all segments of each 7-segment character display light.

Keypad:

The URDA® SDK-86 uses a conductive foam keypad with a rated life of 1,000,000 presses per key. If the SDK-86 is used in an environment with a lot of dust, it may be necessary to clean or replace the rubber portion of the keypad area.

To remove the keypad, simply unscrew the six 4-40 screws and nuts attaching the plastic bezel and the keypad to the PCB board. Then, remove the plastic bezel which covers the keypad and finally the keypad itself from the board.

To clean, wipe the area of the printed circuit board directly underneath the keypad with a clean cloth. Gently brush the round conductive foam pads underneath the keypad with a soft clean brush.

If necessary, a new keypad may be purchased by contacting URDA, Inc.

Replace the rubber keypad being careful to line up the six holes. Gently, replace the plastic bezel, and finally carefully replace the six screws and nuts, being careful not to overtighten the nuts.

CAREFUL: Overtightening the screws could result in a bent plastic bezel and/or deformed plastic bezel-keypad combination.

Conductive foam keypads work by the conductive portion shorting the traces under each key pad on the printed circuit board together. A single firm press on any key will give the same result as a mechanical key except that the conductive foam keypad will have a longer useful life.

WHAT IF IT DOESN'T?

The URDA® SDK-86 was assembled and tested for proper operation after assembly. However, if, after you have configured the SDK-86 as you wish it to be, if there is no response to the RESET Button being pushed,

- ☐ Use a multimeter to check for the presence and proper polarity of +5 Volts on the board.
- ☐ Check all of the jumper connections, and be sure they are in the right places for the configuration you chose.

Effective troubleshooting can be accomplished if you have a dual-trace oscilloscope of at least 20 MHz bandwidth, or a logic analyzer.

- ☐ Pin 19 of cpu A17 (8086) should show a clock output of 2.5 MHz (W40). If it doesn't, there is something wrong with the clock circuitry.
- ☐ Pin 18 of N5 and N6 should have a negative-going pulse about 13 μ s every 15 μ s. This is the Chip Select pulse that indicates that the cpu is fetching instructions.

- ☐ Pin 6 of address decoder N10 (74LS20) should have a negative going pulse of 2 μ s every 15 μ s pulse. If not, the Monitor is not executing.
- ☐ If all else fails, call the URDA® Service Hotline and describe the results of the foregoing procedure.

The telephone number is 1-800-338-0517.

Note: The URDA® Service Hotline is available to provide limited support to help you get your SDK-86 running. If we can't help you over the phone, you may be directed to return your SDK-86 to us and we will repair it for a flat fee and send it back to you. The URDA® Service Hotline is available Monday through Friday, between 9 AM and 5:00 PM Eastern Time.

IMPORTANT: The URDA® Service Hotline *is not able* to provide help to you in writing programs for your SDK-86 or in making hardware modifications. Please rely on the documentation provided with your kit for assistance.



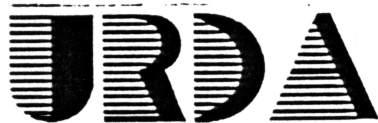
CONTENTS

	PAGE		PAGE
CHAPTER 1			
GENERAL INFORMATION			
Introduction.....	1-1	Move Command	4-7
Specifications.....	1-1	Port Input Command	4-8
CHAPTER 2		Port Output Command	4-9
FUNCTIONAL DESCRIPTION		Go Command	4-10
Introduction.....	2-1	Single Step Command.....	4-11
Clock Generator	2-1	Read Hex File Command	4-12
Wait State Generator	2-2	Write Hex File Command	4-13
CPU	2-3	CHAPTER 5	
Parallel I/O Ports	2-3	SERIAL LOADER	
RAM.....	2-4	Introduction.....	5-1
PROM	2-5	System Operation	5-1
I/O Decode	2-6	Loader Commands	5-2
Off Board Decode.....	2-7	Transfer Hex File Command	5-2
Keypad/Display	2-7	Load Hex File Command	5-3
Serial Interface	2-8	Exit Command	5-4
Bus Expansion.....	2-10	System I/O Routines.....	5-4
CHAPTER 3		Console Input Routine	5-4
KEYPAD MONITOR		Console Output Routine.....	5-6
Introduction.....	3-1	Reader Input Routine	5-8
Keypad	3-1	Punch Output Routine	5-9
Display	3-3	CHAPTER 6	
General Operation.....	3-4	PERIPHERAL INTERFACING	
Monitor Commands	3-6	Introduction.....	6-1
Examine Byte and Examine Word Commands	3-7	Serial Interface Port	6-1
Examine Register Command	3-11	SDK-C86 Interface	6-2
Input Byte and Input Word Commands.....	3-13	Parallel I/O Ports	6-3
Output Byte and Output Word Commands.....	3-15	Bus Expansion Interface.....	6-4
Go Command	3-17	APPENDIX A	
Move Command	3-20	TELETYPEWRITER MODIFICATIONS	
Step Command	3-22	APPENDIX B	
Program Listing	3-24	ROM/EPROM MEMORY	
CHAPTER 4		EXPANSION	
SERIAL MONITOR			
Introduction.....	4-1		
Monitor Command Structure	4-2		
Monitor Commands	4-2		
Substitute Memory Command.....	4-3		
Examine/Modify Register Command.....	4-5		
Display Memory Command.....	4-6		



TABLES

TABLE	TITLE	PAGE	TABLE	TITLE	PAGE
2-1	Wait State Selection	2-2	3-5	Monitor Command Summary	3-6
2-2	Parallel I/O Port Addresses	2-3	3-6	8086 Registers	3-11
2-3	PROM A29 Decoding	2-4	3-7	Parallel I/O Port Addressing	3-14
2-4	PROM A26 Decoding	2-5	3-8	Control Port Addressing	3-16
2-5	PROM A22 Decoding	2-6	4-1	Monitor Command Summary	4-3
2-6	I/O Port Addresses Assignments	2-6	4-2	Register Abbreviations	4-5
2-7	PROM A12 Decoding	2-7	4-3	Parallel I/O Port Addressing	4-8
2-8	Keypad/Display I/O Ports	2-7	4-4	Control Port Addressing	4-9
2-9	Segment Definition	2-8	5-1	Serial Loader Command Summary	5-2
2-10	Baud Rate Selection	2-9	6-1	Serial Interface Connector J7 Pin Assignments	6-1
2-11	USART I/O Ports	2-9	6-2	Parallel I/O Port Connector J5 Pin Assignments	6-3
2-12	Serial Port Jumper Matrix	2-10	6-3	Parallel I/O Port Connector J6 Pin Assignments	6-3
3-1	Hexadecimal Keypad Legend Interpretation	3-2	6-4	Bus Expansion Pin Assignments	6-4
3-2	Function Key Operation	3-3			
3-3	Hexadecimal Display Characters	3-4			
3-4	Register Initialization	3-5			



ILLUSTRATIONS

FIGURE	TITLE	PAGE	FIGURE	TITLE	PAGE
2-1	Monitor Memory Allocation	2-5	3-4	Parallel I/O Port Configuration	3-13
2-2	Detailed Block Diagram	2-11	4-1	Reserved Memory	4-1
3-1	Keypad Arrangement	3-1	6-1	Inteltec Microcomputer Development System Interfacing	6-2
3-2	Reserved Memory	3-5			
3-3	Physical Address Calculation	3-6			



CHAPTER 1 GENERAL INFORMATION

1-1. Introduction

This is the user's manual for the SDK-86. Now that you have completed the initial checkout of your SDK-86, this manual will provide insight into the operation and function of the SDK-86 and also will explore circuit designs that can be implemented either off-board or within the user design area to increase or expand your SDK-86's capabilities. Included within this guide are:

- A description of the SDK-86 circuitry.
- Functional descriptions of the two ROM-resident monitor programs with command examples.
- A description of the "loader" program that is used to perform upload/download program transfers between SDK-86 memory and an Intel microcomputer development system.
- Interfacing and board configuration information for connecting peripheral devices to the SDK-86.
- Appendices describing circuit designs that can be implemented in the user design area including program memory expansion using Intel 2708, 2716 and 2732 EPROM devices.

1-2. Specifications

Central Processor

CPU: 8086
Clock Frequency: 2.5 MHz or 5 MHz (jumper selectable)
Instruction Cycle Time: 800 ns (5 MHz)

Memory Type

ROM: 8K bytes 2732A
RAM: 16K bytes 6264

Memory Addressing

ROM: FE000 through FFFFF
RAM: 0 through 3FFF

Input/Output

Parallel: 48 lines (two 8255A's)
Serial: RS232 or current loop (8251A)
Baud rate selectable from 110 to 4800 baud

Interface Signals

CPU Bus: All signals TTL compatible
Parallel I/O: All signals TTL compatible
Serial I/O: 20 mA current loop or RS232

Interrupts

External: Maskable and Non-Maskable, Interrupt vector 2 reserved for non-maskable interrupt (NMI).

Internal: Interrupt vectors 1 (single-step) and 3 (breakpoint) reserved by monitor.

DMA

Hold Request: Jumper selectable, TTL compatible input.

Software

System Monitors: Preprogrammed 2732A ROMs

Addresses: FE000 through FFFFF

Monitor I/O: Keypad and Serial (teletypewriter or CRT)

Literature Supplied

- SDK-86 User's Manual
- SDK-86 Monitor Listing
- The 8086 Family User's Manual

Physical Characteristics

Length: 34.3 cm (13.5 in.)

Width: 30.5 cm (12.0 in.)

Height: 4.4 cm (1.75 in.)

Weight: Approximately 0.7 kg (1.5 lbs)

Power Requirements

V_{CC} : +5 volts ($\pm 5\%$), 3.5 Amperes

V_{TTY} : -12 volts ($\pm 10\%$), 0.3 Amperes (required only if teletypewriter or CRT terminal connected to serial interface port)

Environmental

Operating Temperature: 0 to 40°C



CHAPTER 2 FUNCTIONAL DESCRIPTION

2-1. Introduction

This chapter examines the circuits which make up the SDK-86. Also presented in this chapter are the I/O port address assignments for the various devices on the board and a definition of ROM and RAM memory allocated to the SDK-86. The individual circuits can be divided into logical blocks as shown in the detailed block diagram (Figure 2-2) at the conclusion of this chapter. Note that the block diagram can be folded out for viewing while reading the circuit descriptions. The individual blocks that will be described in this chapter are as follows:

- Clock Generator
- Wait State Generator
- CPU
- Parallel I/O Ports
- RAM
- PROM
- I/O Decode
- Off Board Decode
- Keypad/Display
- Serial Interface
- Bus Expansion

2-2. Clock Generator

The clock generator circuit is an Intel 8284 clock generator/driver. The circuit accepts a crystal input which operates at a fundamental frequency of 14.7456 MHz. (14.7456 MHz was selected since this frequency is a multiple of the baud rate clock and also provides a suitable frequency for the CPU.) The clock generator/driver divides the crystal frequency by three to produce the 4.9 MHz CLK signal required by the CPU. Additionally, the clock generator performs a further divide-by-two output called PCLK (peripheral clock) which is the primary clock signal used by the remainder of the circuits.

The clock generator/driver provides two control signal outputs which are synchronized (internally) to the 4.9 MHz CLK signal; RDY (ready) and RST (reset). RST is used to reset the SDK-86 to an initialized state and occurs when the $\overline{\text{RES}}$ input goes low (when power first is applied or when the **SYSTEM RESET** key is pressed). The RDY output is active (logically high) when the RDY1 input from the wait state generator is active. As will be explained in the next section, the RDY1 input is active whenever on board memory is addressed or when a selected number of "wait states" occurs.

A jumper link (W40/W41) is provided for low-speed (2.45 MHz) operation and allows the PCLK signal to be routed to the CLK input of the CPU in place of the CLK output from the clock generator/driver. Note that this is the clock configuration that was selected when the SDK-86 kit was assembled and that this configura-

tion allows on-board memory and I/O operations to be performed without the necessity of wait states. When 8086 operation at the 5 (4.9) MHz rate is desired, simply remove the shorting plug from W40 and install it at W41. For additional information regarding the operation of the 8284 clock generator/driver, refer to *The 8086 Family User's Manual*.

NOTE

The SDK-86 may be supplied with an 8086 CPU that has a maximum clock input frequency of 4 MHz and therefore only can be operated with the 2.45 MHz clock (shorting plug installed at W40). These 4 MHz versions of the 8086 are noted by a "-4" suffix following the circuit number stamped on the top of the component (i.e. 8086-4).

2-3. Wait State Generator

The wait state generator circuit allows wait states to be inserted into the CPU's bus cycle to compensate for a "slow" peripheral I/O or memory circuit that has been interfaced to the expansion bus or to allow on-board memory and I/O operations to function correctly when the CPU is operated at a 5 MHz rate. (The SDK-86 memory and I/O devices do not require wait states when the 2.45 MHz clock is used and require 1 wait state when the 4.9 MHz clock is used.) When one or more wait states are required for proper CPU operation, the shorting plug originally installed at W27 (0 wait states) is removed and is positioned over the header pins corresponding to the number of wait states to be inserted into the bus cycle according to the following table.

Table 2-1. Wait State Selection

Plug Position	Wait States	Plug Position	Wait States
W27*	0	W31	4
W28	1	W32	5
W29	2	W33	6
W30	3	W34	7

*Omitting the wait state shorting plug is equivalent to 0 wait states.

Referring to the block diagram (Figure 2-2), the wait state generator is cleared following every read, write or interrupt cycle and subsequently is enabled at the beginning of the next read, write or interrupt cycle when the CLR input to the wait state generator goes inactive. When enabled, the wait state generator (a 74LS164 shift register) begins to shift a "one" through the register. When the selected number (0-7) of shifts have occurred, the jumpered output goes active which, coupled to the RDY1 input of the clock generator/driver, causes the clock generator's RDY output to the CPU to go active.

When a wait state is selected, on-board I/O operations are subject to the number of wait states selected, while on-board memory operations are performed without the wait state restriction. As shown on the block diagram, the RDY1 input to the clock generator/driver originates either from the wait state generator or from a three-input AND gate. The output from the gate is active during on-board memory read and write operations (M/ $\overline{\text{IO}}$ and $\overline{\text{OE}}$ BOARD high). The jumper link (W39) at one input to the AND gate, when installed, permanently disables the gate and causes on-board memory operations to be subject to the number of wait states selected. Note that it may be necessary to install this link if "slow" ROM or RAM on-board memory is added.

2-4. CPU

Information on the operation, function and instruction set of the 8086 CPU is contained in *The 8086 Family User's Manual* and is not repeated in this publication. Several points regarding the implementation of the 8086 CPU in the SDK-86 should be noted.

- The 8086 is used in the minimum mode (MN/ \overline{MX} input held logically high).
- The INTR, TEST and HOLD inputs to the 8086 were disabled by the shorting plugs installed in header W36-W38 when the kit was assembled. When a peripheral circuit is interfaced to the bus expansion logic and requires the use of any of these signals, the corresponding shorting plugs must be removed.
- When the SDK-86 is reset, the 8086 executes the instruction at location FFFF0H. The instruction at this location is an inter-segment direct jump to the beginning of the monitor program that resides in ROM locations FF000H to FFFFFH.
- The 8086's NMI (non-maskable interrupt) input originates from the INTR key and, when pressed, causes the CPU to save the current system status (pushing the IP, CS and FL register contents onto the stack after clearing the IF and TF flags) and to perform an indirect long jump through RAM location 08H (interrupt vector 2). Note that locations 08H through 0BH are initialized by the monitor on reset to contain the address of the monitor's interrupt routine.
- The maskable interrupt (INTR) is not used by the SDK-86, but is available to peripheral circuits through the expansion bus. To use the maskable interrupt, an interrupt vector pointer must be provided on the data bus when \overline{INTA} is active. When there is more than one source for an interrupt, an interrupt priority resolution circuit must be provided (e.g., an Intel 8259A).

2-5. Parallel I/O Ports

The parallel I/O ports consist of two Intel 8255A programmable peripheral interface circuits. These two circuits each contain three 8-bit input/output data ports (designated ports A, B and C) and one write-only control port. The 8255A circuit that interfaces with the low-order data byte (D0-D7) is designated port 2 (P2), and the 8255A circuit that interfaces with the high-order data byte (D8-D15) is designated port 1 (P1). All ports can be addressed individually (e.g., port P1A, P2C, P1 Control, etc.) or a corresponding pair of ports (P1A and P2A, P1B and P2B or P1C and P2C) can be addressed simultaneously to form a single 16-bit wide data port.

The I/O port address assignments for the two parallel I/O port circuits are defined in the following table.

Table 2-2. Parallel I/O Port Addresses

Port	Address	Port	Address
P2A	FFF8	P1A	FFF9
P2B	FFFA	P1B	FFFB
P2C	FFFC	P1C	FFFD
P2 Control	FFFE	P1 Control	FFFF

During byte operations, the I/O decode logic generates the appropriate \overline{CS} (chip select) input (HIGH PORT SELECT or LOW PORT SELECT), while during word operations, the P2 port address is used to address the desired pair of ports, and the

I/O decode logic generates both HIGH PORT SELECT and LOW PORT SELECT coincidentally. More detailed information on the operation and programming of the 8255A can be found in the *Intel Component Data Catalog*.

2-6. RAM

The standard URDA® SDK-86 includes 16K bytes of random access memory (RAM). RAM is located at the low end of memory (locations) 0H through 3FFFH). The memory addressing arrangement of the 8086 CPU allows simultaneous reading or writing of a full 16 bits (two adjacent byte locations) or the reading or writing of either the high (D8-D15) or low (D0-D7) the byte.

RAM is enabled by the appropriate outputs(s) from RAM decode circuitry and by a high level on the M/I O control line (memory operation). The PROMs N5 and N6 are enabled by decoding BHE and the A0 and A13 and A19 address bits. The A0 address bit and BHE determine the byte or bytes enabled. The following table defines the PROM decoding.

Table 2-3. RAM Decode

Logic Inputs			Logic Outputs		Byte(s) Selected
A14-A19	BHE	A0	\overline{CE} (EVEN)	\overline{CE} (ODD)	(Address Block)
0	0	0	0	0	Both Bytes (0H-3FFFH)
0	0	1	1	0	Odd Bytes (0H-3FFFH)
0	1	0	0	1	Even Byte (0H-3FFFH)
0	1	1	1	1	None

RAM itself is addressed by the A1 through A13 address bits and is controlled (read or write) by the R D and WR signals.

The first 256 (decimal) byte locations of RAM (locations 0H through 0FFH) are reserved by the monitor program. The following illustration shows the actual monitor allocations within this block

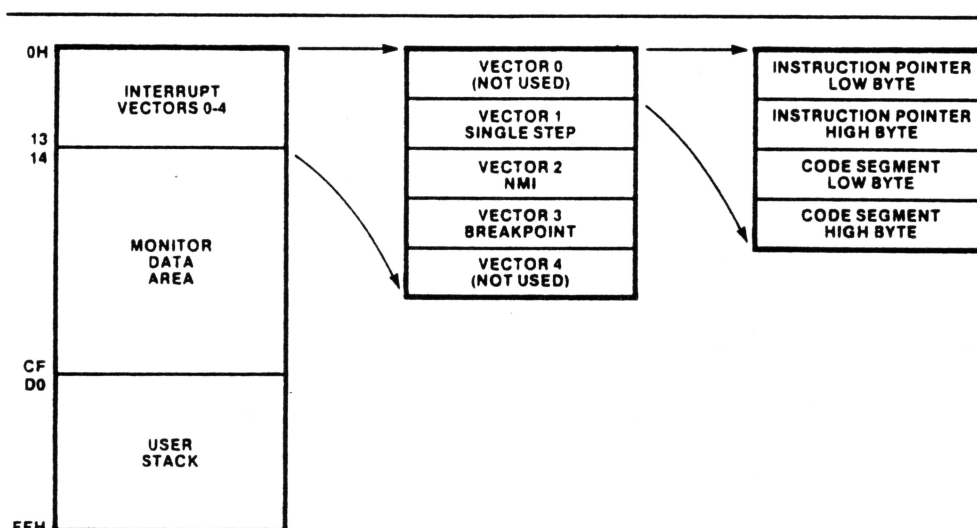


Figure 2-1. Monitor Memory Allocation

2-7. PROM

The URDA® SDK-86 includes 8K bytes of EPROM containing the keypad monitor program and the serial monitor program. The 8K bytes of EPROM are located at the high end of memory (locations FE000H through FFFFFH). The principal system monitor program (keypad or serial) resides in locations FF000H through FFFFFH (sockets N5 and N6). The remaining 4K (locations FE000H through FEFFFH) are used for the serial monitor program.

EPROM is enabled by the corresponding output from decode circuitry and by the BHE and address bit A0 inputs. The decode circuitry decodes the M/I O signal and the A12 through A19 address bits to determine which 4K bytes (FE000H-FEFFFH or FF000H-FFFFFH) are to be addressed, while the BHE and A0 inputs enable either high- or low-order byte or word (both high and low bytes) output.

Referring to sheet 1 of the SDK-86 schematic diagram, two additional enable outputs (not shown on the block diagram) are provided. These outputs (CSX and CSY) are routed to bus expansion connector J2 to permit two additional 4K bytes of PROM or ROM (at locations FC000H-FCFFFH and FD000H-FDFFFH) to be incorporated without requiring external address decoding logic.

Table 2-4 defines the EPROM's address decoding.

Table 2-4. EPROM Decode

Logic Inputs					Logic Outputs				Byte(s) Selected
A14-A19	A13	A12	BHE	A0	CE(EVEN)	CE(ODD)	CSX	CSY	(Address Block)
1	1	x	0	0	0	0	1	1	Both Bytes (FE000H-FFFFFH)
1	1	x	0	1	1	0	1	1	Odd Byte (FE000H-FFFFFH)
1	1	x	1	0	0	1	1	1	Even Byte (FE000H-FFFFFH)
1	1	x	1	1	1	1	1	1	None
1	0	0	x	x	1	1	1	0	FC000H-FCFFFH (CSY)
1	0	1	x	x	1	1	0	1	FD000H-FDFFFH (CSX)

2-8. I/O Decode

I/O decode circuitry is enabled by a low level $\overline{M/\overline{IO}}$ input (indicating an I/O operation) and an address between FE00H and FFFFH (address bits A11 through A15 active). When enabled, the circuitry decodes BHE, A0 and A3 through A10 address bits to generate the corresponding I/O port select signal(s) according to Table 2-5.

Table 2-5. I/O Decode

Inputs					Outputs			
A5-A10	A4	A3	\overline{BHE}	A0	$\overline{HIGH\ PORT\ SELECT}$	$\overline{LOW\ PORT\ SELECT}$	$\overline{USART\ SELECT}$	\overline{KSEL}
1	0	1	0	0	1	1	1	0
1	0	1	1	0	1	1	1	0
1	1	0	0	0	1	1	0	1
1	1	0	1	0	1	1	0	1
1	1	1	0	0	0	0	1	1
1	1	1	0	1	0	1	1	1
1	1	1	1	0	1	0	1	1
All other states					1	1	1	1

2-9. Off Board Decode

The off board decode logic is responsible for the generation of the OFF BOARD signal. This signal is used by the wait state generator to force off-board memory operations to be subjected to the number of wait states selected and also is responsible for the generation of the BUFFER ON signal that is used to enable the data transceivers on the expansion bus during both I/O operations and off-board memory accesses.

Off board decode circuitry accepts the A13 through A19 address bits and the M/IO signal to generate the OFF BOARD signal when off-board memory locations are addressed. Table 2-7 defines the decoding Note that the table shows the inactive states in which the circuitry *does not* generate the OFF BOARD signal.

Table 2-7. Off Board Decode

xx in Axx																			Output	Corresponding Address Block
19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0H-3FFFH (On Board RAM)
1	1	1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	1	FE00H-FFFFH (On Board EPROM)
																			0	4000H-FDFFFH (Off Board)

The OFF BOARD signal also is generated when an off-board I/O port is addressed (port addresses 0H through 0FFDFH). During an I/O operation, OFF BOARD the gating above is enabled by the low state M/IO and generates the OFF BOARD signal if any of the A5 through A15 address bits is at a low state (an address below 0FFE0H).

2-10. Keypad/Display

The keypad/display logic is centered around an Intel 8279 Keyboard/Display Controller. The 8279 is responsible for the debouncing of the keys, the coding of keypad matrix and the refreshing of the display elements.

As stated in Section 2-8, the 8279 occupies two I/O ports within the on-board I/O address space and, since the 8279 is interfaced to the lower byte of the data bus (D0-D7), both ports have even-numbered port addresses (FFE8H and FFEAH). The individual 8279 port functions are determined by the A1 address bit and the RD and WR signals as noted in Table 2-8.

Table 2-8. Keypad/Display I/O Ports

8279 Input			Port Address	Port Function
A1	RD	WR		
0	0	1	FFE8H	Read Display RAM or Keyboard FIFO
0	1	0	FFE8H	Write Display RAM
1	0	1	FFEAH	Read Status
1	1	0	FFEAH	Write Command

Since the A2 address bit is not decoded by the I/O decode circuitry, port addresses FFECH and FFEH are decoded as port addresses FFE8H and FFEAH, respectively. As noted in Table 2-6, these two additional addresses are reserved.

The monitor program, through the write command port, arranges the 8279 as follows:


- Eight digit, 8-bit, left entry display and encoded scan keyboard with 2-key lockout (keyboard/display mode set command; byte value 00H).
- A prescale factor of 25 to provide 5 ms keyboard and display scan times and a 10 ms debounce time (program clock command; byte value 039H).

Additional information regarding the operation and programming of the 8279 can be found in the *Intel Component Data Catalog*.

Referring again to the block diagram, since the encoded scan keyboard mode is specified, the SL0 through SL3 outputs represent a binary count. The 7445 (a BCD to 10-line decoder) converts the binary count to eight, single-line outputs (two outputs are unused) which are used to enable the individual display elements. The SL0 through SL2 outputs additionally are routed to a 74LS156 (configured as a 3- to 8-line decoder) to provide the three row scan signal inputs to the keypad switch matrix. The outputs from the switch matrix, at the RL0 through RL7 inputs to the 8279, represent the eight switch columns (see sheet 7 of the SDK-86 schematic). When a switch is pressed while its row is being scanned, the corresponding column is enabled. The 8279 uses the enabled column bit value and its row scan value (SL0-SL2) to generate a 6-bit code representing the key pressed. This is the code that is stored in the FIFO and accessible to the CPU through the read keyboard FIFO I/O port.

The A0 through A3 and B0 through B3 outputs from the 8279 form a single 8-bit parallel output containing the individual segment enable bits. Table 2-9 defines the correlation between the individual bits and the display element segments (a high level output turns on the corresponding display segment).

Table 2-9. Segment Definition

 Display Segments	8279 Output Bit	Segment Enabled	8279 Output Bit	Segment Enabled
	A0	e	B0	a
	A1	f	B1	b
	A2	g	B2	c
	A3	DP	B3	d

2-11. Serial Interface

The serial interface is based on an Intel 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter) which is operated in the asynchronous mode. An associated baud rate generator provides jumper-selectable baud rates ranging from 75 to 4800 baud for use by the USART. The serial port jumper matrix is used to configure the USART's input/output signals at serial interface connector J7 for "stand alone" or "MDS slave" operation with either teletypewriter (20 mA current loop) or CRT terminal (RS232) interface capability.

The monitor program, by writing a mode command of 0CFH to the USART's control port, configures the USART for:

- 8-bit character length
- Parity disabled
- Two stop bits
- Baud rate factor of 64x

The baud rate generator (a 74LS393 dual 4-bit binary counter) uses the 614.4 kHz signal (PCLK/4) to provide a series of baud rate frequencies. Since the USART is operated in the 64x mode, these frequencies are 64 times the corresponding baud rate. Table 2-10 defines the baud rate selection and corresponding baud rate generator output frequency.

Table 2-10. Baud Rate Selection

Baud Rate	Shorting Plug Position	Output Frequency
4800	W25	307.2 kHz
2400	W24	153.6 kHz
1200	W23	76.8 kHz
600	W22	38.4 kHz
300	W21	19.2 kHz
150	W20	9.6 kHz
75	W19	4.8 kHz
110	W20, W26	6.98 kHz

Referring to sheet 9 of the SDK-86 schematic, the 110 baud rate is derived from the 76.8 kHz signal at the 2A input to the second binary counter. When the shorting plug at W26 is installed, the NAND gate on the 2QA, 2QB and 2QD outputs clears the second binary counter on the count of eleven to provide the required 6.98 kHz signal at the 2QC output.

The USART occupies two I/O ports within the on-board I/O address space and, since the USART is interfaced to the lower byte of the data bus (D0-D7), both ports have even-numbered port addresses (FFF0H and FFF2H). The individual USART port functions are determined by the A1 address bit and the \overline{RD} and \overline{WR} signals as noted in Table 2-11.

Table 2-11. USART I/O Ports

USART Input			Port Addresses	Port Function
A1	\overline{RD}	\overline{WR}		
0	0	1	FFF0H	Read USART Data
0	1	0	FFF0H	Write USART Data
1	0	1	FFF2H	Read USART Status
1	1	0	FFF2H	Write USART Control

Since the A2 address bit is not decoded by the I/O decode circuitry, port addresses FFF4H and FFF6H are decoded as port addresses FFF0H and FFF2H, respectively. As noted in Table 2-6, these two additional addresses are reserved.

The serial port jumper matrix determines the individual pin assignments of the USART's input/output signals at connector J7 and the signal definition (current loop or RS232) and polarity. Table 2-12 defines the shorting plugs to be installed for the various interface configurations.

Table 2-12. Serial Port Jumper Matrix

Interface Configuration	Shorting Plugs Installed	PC Board Silkscreen
STAND ALONE CRT TERMINAL	W1-W5	└ CRT ┘
STAND ALONE TELETYPEWRITER	W8-W16	└ TTY ┘
INTELLEC SLAVE CRT TERMINAL	W3-W7	└ MDS- CRT ┘
INTELLEC SLAVE TELETYPEWRITER	W14-W18	└ MDS- TTY ┘

Additional information regarding the operation and programming of the 8251A USART can be found in the *Intel Component Data Catalog*.

2-12. Bus Expansion

The bus expansion logic allows peripheral circuits to be interfaced to the SDK-86. The bus expansion logic, which consists of the control transceivers and drivers, the data transceivers and the address latches, provides all required address, data and control signals for either on-board (user design area) or off-board interfacing of peripheral circuits connected to bus expansion connectors J1/J2 and J3/J4.

The control transceivers (an Intel 8286 bidirection bus driver) determine the source of the five bidirection control signals. When a peripheral "master" device is granted bus control by the 8086 CPU, the $\overline{\text{HLDA}}$ signal at the transceiver's transmit (T) input is active and causes the transceivers to function as receivers (origin of the control signals is from the peripheral circuit). Conversely, while the 8086 has control of the bus, $\overline{\text{HLDA}}$ is inactive, and the control signals originate from the 8086 CPU.

The (control) drivers (a 74LS244 three-state latch) enable the associated 8086 CPU control and status signals onto the expansion bus while the 8086 has control of the bus ($\overline{\text{HLDA}}$ inactive) and, when bus control is granted to the peripheral circuit ($\overline{\text{HLDA}}$ active), the drivers are placed in their high-impedance state.

The data-transceivers (two Intel 8286 bidirectional bus drivers) determine the source of the high- and low-order data bytes during off-board memory and I/O operations and interrupt acknowledge cycles. Referring to the block diagram, the transceivers are enabled by the $\overline{\text{BUFFER ON}}$ signal. This signal is active during the T_2 to T_4 instruction cycles ($\overline{\text{DEN}}$ active) for an off-board memory and I/O operation ($\overline{\text{OFF BOARD}}$ active) or an interrupt acknowledge cycle ($\overline{\text{INTA}}$ active). When the transceivers are enabled, the state of the $\text{DT}/\overline{\text{R}}$ signal at the T input determines the direction of the transceivers.

The expansion bus address lines (and the $\overline{\text{BHE}}$ signal) originate directly from the address latches (three 74S373 three-state latches) when the 8086 has control of the bus ($\overline{\text{HLDA}}$ inactive). When bus control is granted to the peripheral device ($\overline{\text{HLDA}}$ active), the latches are placed in their high-impedance state.

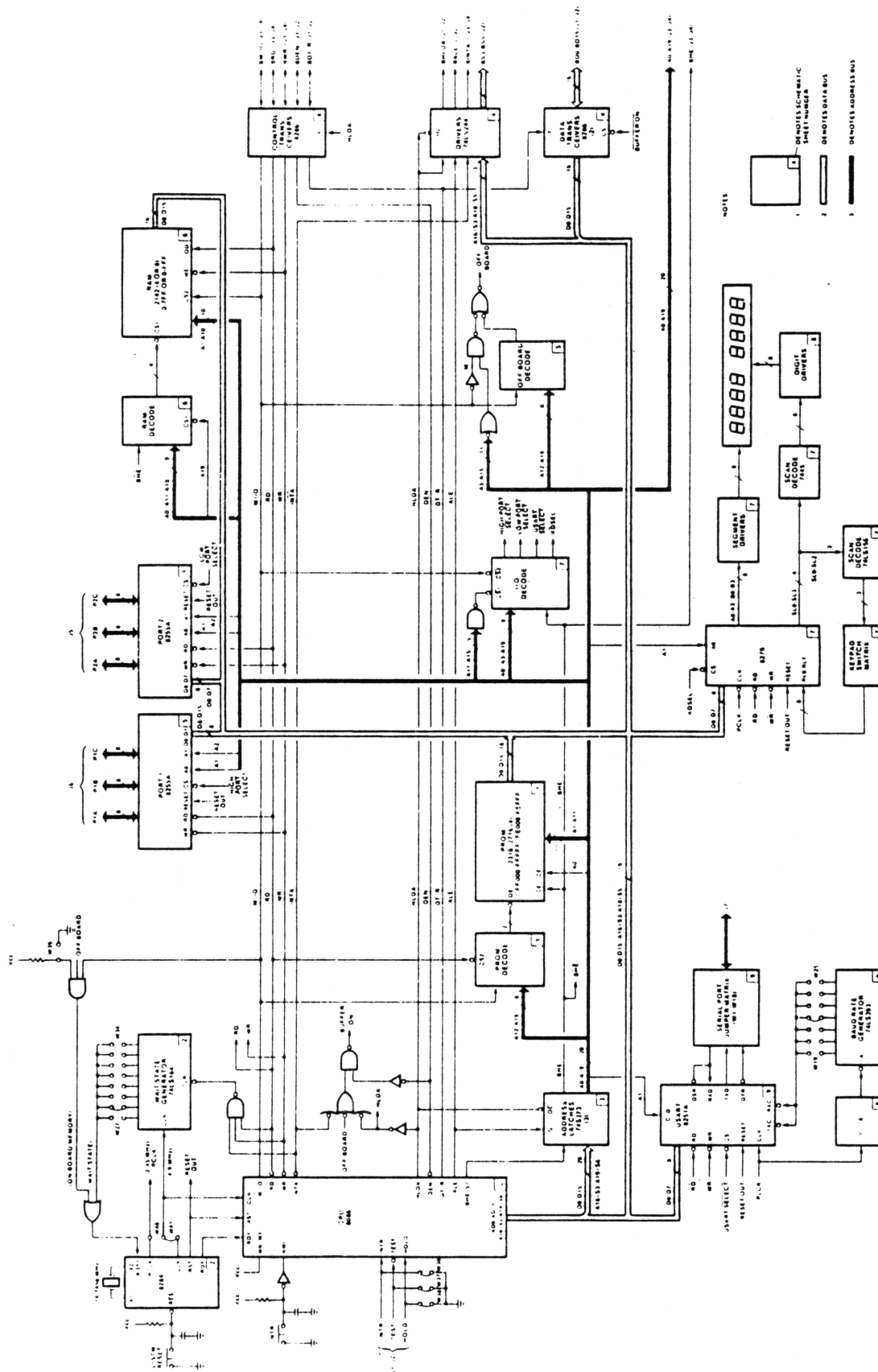


Figure 2-2. Detailed Block Diagram



3-1. Introduction

This chapter describes your "interaction" or "how you communicate" with your MCS-86 System Design Kit through the keypad monitor program. The monitor program resides in 4K bytes of ROM (Read Only Memory) at the upper end of memory. The program is "initialized" or ready whenever power is turned on or any time the **SYSTEM RESET** key is pressed and allows you to perform the following operations using the keypad and display.

- Examine and modify registers within the 8086 microprocessor.
- Examine and modify memory locations.
- Enter and initiate execution of your own programs or subroutines.
- Evaluate execution (debug) of your program through the monitor's single-step and breakpoint facilities.
- Move selected blocks of memory from one location to another.
- Write or read data to or from an I/O port.

3-2. Keypad

With the keypad monitor program, you enter both commands and data by pressing individual keys of the keypad. (The monitor communicates with you through the display.) As shown in Figure 3-1, the keypad is divided into two logical groups; the 16 hexadecimal keys on the right-hand side and the eight function keys on the left-hand side.

Most of the hexadecimal keys have combined functions as noted by their individual legends. The small letters appearing under the hexadecimal key values are acronyms for individual monitor commands and 8086 register names. Acronyms to the left of the slash sign are monitor commands, and acronyms to the right of the slash sign are 8086 register names. The function of a hexadecimal key at any one time is dependent on the current state of the monitor and what the monitor is expecting as input. Table 3-1 defines both the commands and registers associated with the hexadecimal keys.

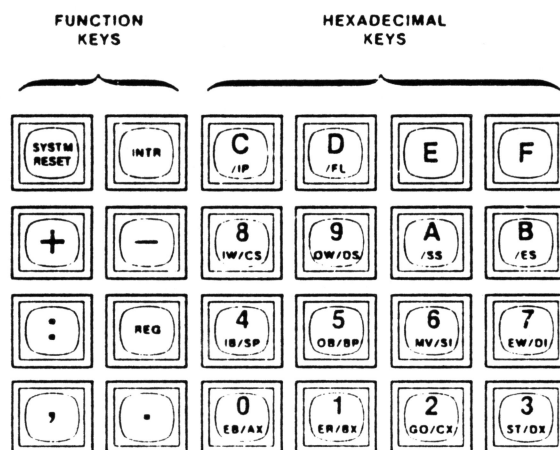










Figure 3-1. Keypad Arrangement

Table 3-1. Hexadecimal Keypad Legend Interpretation

Hexadecimal Key	Command		Register	
	Acronym	Name	Acronym	Name
0 EB/AX	EB	Examine Byte	AX	Accumulator
1 ER/BX	ER	Examine Register	BX	Base
2 GO/CX	GO	Go	CX	Count
3 ST/DX	ST	(Single) Step	DX	Data
4 IB/SP	IB	Input Byte	SP	Stack Pointer
5 OB/BP	OB	Output Byte	BP	Base Pointer
6 MV/SI	MV	Move	SI	Source Index
7 EW/DI	EW	Examine Word	DI	Destination Index
8 IW/CS	IW	Input Word	CS	Code Segment
9 OW/DS	OW	Output Word	DS	Data Segment
A /SS	none	N/A	SS	Stack Segment
B /ES	none	N/A	ES	Extra Segment
C /IP	none	N/A	IP	Instruction Pointer
D /FL	none	N/A	FL	Flag
E	none	N/A	none	N/A
F	none	N/A	none	N/A

The individual operation of the eight function keys is defined in Table 3-2.

Table 3-2. Function Key Operation

Function Key	Operation
	The SYSTM RESET key allows you to terminate any present activity and to return your SDK-86 to an initialized state. When pressed, the 8086 sign-on message appears in the display and the monitor is ready for command entry.
	The INTR (interrupt) key is used to generate an immediate, non-maskable type 2 interrupt (NMI). The NMI interrupt vector is initialized on power up or system reset to point to a routine within the monitor which causes all of the 8086's registers to be saved. Control is returned to the monitor for subsequent command entry.
	The + (plus) key allows you to add two hexadecimal values. This function simplifies relative addressing by allowing you to readily calculate an address location relative to a base address.
	The - (minus) key allows you to subtract one hexadecimal value from another.
	The : (colon) key is used to separate an address to be entered into two parts; a segment value and an offset value.
	The REG (register) key allows you to use the contents of any of the 8086's registers as an address or data entry.
	The , (comma) key is used to separate keypad entries and to increment the address field to the next consecutive memory location.
	The . (period) key is the command terminator. When pressed, the current command is executed. Note that when using the Go command, the 8086 begins program execution at the address specified when the key is pressed.

3-3. Display

Your SDK-86 uses the eight-digit display to communicate with you. Depending on the current state of the monitor, the information displayed will be the:

- Current contents of a register or memory location
- An "echo" of a hexadecimal key entry
- A monitor prompt sign
- An information or status message

The display itself is divided into two groups of four characters. The group on the left is referred to as the "address field," and the group on the right is referred to as the "data field." All values displayed are in hexadecimal and follow the format shown in Table 3-3.

Table 3-3. Hexadecimal Display Characters

Hexadecimal Value	Display Format
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	A
B	b
C	C
D	d
E	E
F	F

3-4. General Operation

When using the keypad monitor, you will be prompted through the display as to the input required. Whenever the monitor is expecting a command entry, a minus sign or "dash" appears in the most significant display digit of the address field. Pressing one of the hexadecimal "command" keys (keys 0-9) when the dash is displayed is interpreted as a command entry. When the key is pressed, the dash disappears and a decimal point (or decimal points) appears in the least significant display digit (or least significant digits) of the address field to indicate that subsequent keypad entry will be directed to the address field. Note that depending on the command, characters also may appear within the address and data fields. Monitor operation from this point is determined by the actual command entered. Refer to Sections 3-6 through 3-12 for individual command format and operation.

Following power-on or whenever the **SYSTM RESET** key is pressed, the monitor initializes the SDK-86 and displays the monitor sign-on message ("86" in the two least-significant digits of the address field and the program's version number in the two least-significant digits of the data field) and the command prompt character (dash) in the most significant digit of the address field. When initialized, the following 8086 registers are set to the values shown in Table 3-4 by the monitor.

Table 3-4. Register Initialization

Register	Value
CS (Code Segment)	0H
DS (Data Segment)	0H
ES (Extra Segment)	0H
SS (Stack Segment)	0H
IP (Instruction Pointer)	0H
FL (Flag)	0H
SP (Stack Pointer)	0100H

Note that in the above table and in the remainder of this manual, the letter "H" is used to indicate that the associated value is in hexadecimal.

The first 256 memory locations (locations 0H-0FFH) are reserved for the monitor and user stack area as shown in Figure 3-2. (First user-available memory location in RAM is 0100H.)

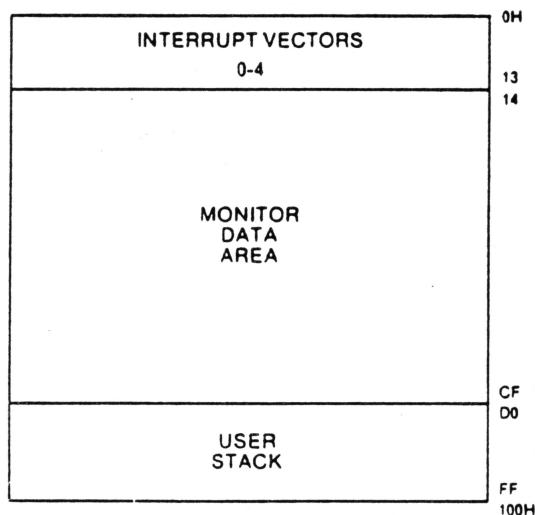


Figure 3-2. Reserved Memory

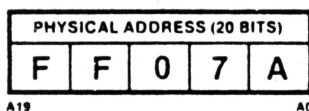
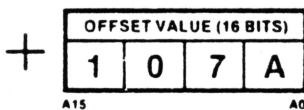
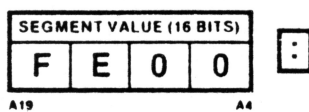
Whenever the SDK is powered up or whenever the **SYSTM RESET** key is pressed, the monitor immediately terminates its present activity and jumps to its initialization routine. This routine initializes interrupt vectors 1 through 3 as follows:

- Interrupt 1: Single Step: Used with the Single Step command
- Interrupt 2: NMI (Non-Maskable Interrupt): Monitors INTR key
- Interrupt 3: Breakpoint: Used with the Go command

Whenever the monitor is re-entered as a result of a Single Step, NMI or Breakpoint interrupt, the monitor temporarily stores (pushes) the contents of the 8086 registers onto the user stack and subsequently removes (pops) the register contents from the stack before it prompts for command entry. Since the SP register is initialized to 0100H (base of the stack), the initial stack reserved for the user is 48 bytes (locations D0-0FFH) of which 26 bytes must be reserved for the register contents should one of the above interrupts occur.

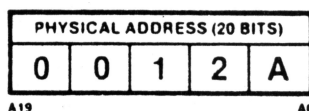
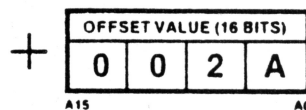
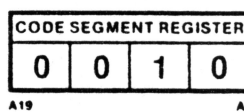
Note that in the following command descriptions, the monitor always calculates a 20-bit physical memory address from a 16-bit segment address value and a 16-bit offset address value. The segment address value is entered first, the ":" key is pressed (to separate the two entries), and then the offset address value is entered. The two values entered, which are displaced from one another by four bits, are added together as shown in Figure 3-3 (Segment-Offset Entry) to form the 20-bit physical memory address. If only one address value is entered (the colon must be omitted), it is interpreted by the monitor as an offset address value entry, and the current contents of the code segment (CS) register are used as the segment address value. The CS register contents and the offset address value entered are combined as shown in the Single Offset Entry example of Figure 3-3 to form the 20-bit physical memory address.

ENTRY: FE00:107A



SEGMENT-OFFSET ENTRY

ENTRY: 2A (CS REGISTER CONTAINS 010H)



SINGLE OFFSET ENTRY

Figure 3-3. Physical Address Calculation

3-5. Monitor Commands

The keypad monitor is capable of executing ten individual commands. Each command is summarized in Table 3-5 and is described, in detail, within the sections which follow. In both the table and the individual command descriptions, the following command syntax is used:

☒ Indicates a keyboard key

[A] Indicates that "A" is optional

[A]* Indicates one or more optional occurrences of "A"

 Indicates that "B" is a variable

Table 3-5. Monitor Command Summary

Command	Function/Syntax
Examine Byte	Displays/modifies memory byte locations EB <addr> [[<data>]] * []
Examine Word	Displays/modifies memory word locations EW <addr> [[<data>]] * []
Examine Register	Displays/modifies 8086 register contents ER <reg key> [[<data>]] * []
Input Byte	Displays data byte at input port IB <port addr> [[]] * []
Input Word	Displays data word at input port IW <port addr> [[]] * []
Output Byte	Outputs data byte to output port OB <port addr> [<data> [[<data>]] * []
Output Word	Outputs data word to output port OW <port addr> [<data> [[<data>]] * []
Go	Transfers control from monitor to user program GO [<addr>] [[<breakpoint addr>]]
Move	Moves block of data within memory MV <start addr> [<end addr> [<destination addr>]]
Step	Executes single user program instruction ST [<start addr>] [[<start addr>]] * []

3-6. Examine Byte and Examine Word Commands

FUNCTION

The Examine Byte (EB) and Examine Word (EW) commands are used to examine the contents of selected memory locations. If the memory location can be modified (e.g., a location in RAM), the contents additionally can be updated.

SYNTAX

EB <addr> [[<data>]] * []

EW <addr> [[<data>]] * []

OPERATION

To use either command, press the **EB** key (examine byte) or **EW** key (examine word) when the command prompt character (-) is displayed. When either key is pressed, the decimal point at the right of the address field will light (the rest of the display will be blanked) to indicate that entry from the keyboard will be directed to the address field. From the keypad, enter the memory address of the byte or word to be examined, most significant character first. Note that all memory addresses consist of both a segment value and an offset value. When no segment value is specified, the default segment value is the current contents of the code segment (CS) register. When a segment value is specified, the first address entry is the segment value, a colon (:) is entered as a separator, and the second address entry is the offset value. The capacity of an address field entry is limited to four characters and, if more than four characters are entered for either a segment or offset value, only the last four

characters entered (the four characters currently displayed) are valid. After the address is entered, press the “,” key. The data byte or word contents of the addressed memory location will be displayed in the data field, and a decimal point will appear at the right of the data field to indicate that any subsequent hexadecimal keypad entry will be directed to the data field. Note that when using the Examine Word command, the byte contents of the memory location displayed appear in the two least-significant digits of the data field, and the byte contents of the next consecutive memory location (memory address + 1) appear in the two most-significant digits of the data field.

If the contents of the memory location addressed only are to be examined, press the “.” key to terminate the command, or press the “,” key to examine the next consecutive memory location (Examine Byte command) or the next two consecutive memory locations (Examine Word command). To modify the contents of an addressed memory location, key-in the new data from the hexadecimal keypad. Note that the data field is limited to either two (examine byte) or four (examine word) characters and that if more characters are entered, only the characters currently displayed in the field are valid. The data displayed is not updated in memory until either the “.” or “,” key is pressed. If the “.” key is pressed, the command is terminated, and the command prompt character is displayed in the address field. If the “,” key is pressed, the *offset* address and data contents of the next consecutive memory location (Examine Byte command) or memory locations (Examine Word command) are displayed.

ERROR CONDITIONS

Attempting to modify a non-existent or read-only (e.g., EPROM) memory location. Note that the error is not detected until the “,” or “.” key is pressed. When an error is detected, the characters “Err” are displayed with the command prompt character in the address field.

EXAMPLES

Example 1: Examining a Series of Memory Byte Locations Relative to the CS Register

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTEM RESET	- 0 0	1 1	System Reset
0 EB/AX			Examine Byte Command
1 ER/BX			} First Memory Location To Be Examined
9 OW/DS	1 9		
,	1 9	x x	Memory Data Contents
,	1 A	x x	Next Memory Location and Data Contents
,	1 b	x x	Next Memory Location and Data Contents
,	1 C	x x	Next Memory Location and Data Contents
.	-		Command Termination/Prompt

Example 2: Examining and Modifying Memory Word Location 10H Relative to the DS Register

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTEM RESET	- 0 0 0	0 0 0 0	System Reset
7 EW/DI	0 0 0 0	0 0 0 0	Examine Word Command
REG	0 0 0 0	0 0 0 0	Register Input
9 OW/DS	0 0 0 0	0 0 0 0	DS Register
:	0 0 0 0	0 0 0 0	Segment/Offset Separator
1 ER/BX	0 0 0 0	0 0 0 0	Offset Address
0 EB/AX	0 0 0 0	0 0 0 0	
,	0 0 0 0	X X X X	Memory Data Contents
8 IW/CS	0 0 0 0	0 0 0 0	New Data to Be Entered
C /IP	0 0 0 0	0 0 0 0	
F	0 0 0 0	0 0 0 0	
B /ES	0 0 0 0	0 0 0 0	
.	- 0 0 0	0 0 0 0	Data Updated, Command Termination/Prompt

To check that the data was updated successfully, press the **EW** key and enter the memory address (DS:10H). Press the “,” key and note that “8CFb” is displayed in the data field.

Example 3: Attempting to Modify PROM

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTEM RESET	- 0 0	1 1	System Reset
0 EB/AX			Examine Byte Command
F			Segment Address
F	F F		
0 EB/AX	F F 0 0		
0 EB/AX	F F 0 0		
:	F F 0 0		Segment/Offset Separator
0 EB/AX			Offset Address
,		9 0	Data Contents of Location FF000H
A		0 A	New Data To Be Entered
7		A 7	
,	- E r r		Error Message

The error message (Err) resulted from trying to modify the data contents of a read only memory location. Repeat the keying sequence to see that the memory contents (90H) of location FF000H were unaltered.

3-7. Examine Register Command

FUNCTION

The Examine Register (ER) command is used to examine and, if desired, to modify the contents of any of the 8086's registers.

SYNTAX

[ER] <reg key>[[<data>]]* []

OPERATION

To examine the contents of a register, press the **ER** key when prompted for a command entry. When the key is pressed, the decimal point at the right of the address field will light. Unlike the Examine Byte command, subsequent hexadecimal keypad entry will be interpreted as the register name (the acronym to the right of the slash sign on the key face) rather than its hexadecimal value. When the hexadecimal key is pressed, the corresponding register abbreviation will be displayed in the address field, the 16-bit contents of the register will be displayed in the data field and the decimal point on the right of the data field will light. Table 3-6 defines the 8086 register name, the hexadecimal keypad acronym, the display abbreviation and the sequence in which the registers are examined.

Table 3-6. 8086 Registers

Register Name	Keypad Acronym	Display Abbreviation
Accumulator	AX	A
Base	BX	b
Count	CX	C
Data	DX	d
Stack Pointer	SP	SP
Base Pointer	BP	bP
Source Index	SI	SI
Destination Index	DI	dI
Code Segment	CS	CS
Data Segment	DS	dS
Stack Segment	SS	SS
Extra Segment	ES	ES
Instruction Pointer	IP	IP
Flag	FL	FL

When the register contents are displayed (when the decimal point on the right of the data field lights), the register's contents can be modified or the register is said to be "open" for input. Keying in a value from the hexadecimal keypad will be echoed in the display's data field, and the register contents will be updated with the data value displayed when either the "," or "." key is pressed. If the "." key is pressed, the command is terminated, and the command prompt character is displayed. If the "," key is pressed, the abbreviation and contents of the "next" register are displayed and opened for modification according to the order in Table 3-6. Note that the sequence is *not* circular and that pressing the "," key when the flag (FL) register is displayed will terminate the examine register command and return to the command prompt character.

EXAMPLES

Example 1: Examining and Modifying a Register

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTM RESET	- 8 6	1 1	System Reset
1 ER/BX			Examine Register Command
8 /ES	E S	0 0 0 0	Extra Segment Register Contents
1 ER/BX	E S	0 0 0 1	New Register Contents
0 EB/AX	E S	0 0 1 0	
.	-		Register Updated, Command Termination/Prompt

Example 2: Examining a Series of Registers

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTM RESET	- 8 6	1 1	System Reset
1 ER/BX			Examine Register Command
9 OW/DS	D S	0 0 0 0	Data Segment Register Contents
,	S S	0 0 0 0	Stack Segment Register Contents
,	E S	0 0 0 0	Extra Segment Register Contents
,	I P	0 0 0 0	Instruction Pointer Register Contents
,	F L	0 0 0 0	Flag Register Contents
,	-		Command Termination/Prompt

3-8. Input Byte and Input Word Commands

FUNCTION

The Input Byte (IB) and Input Word (IW) commands are used to input (accept) an 8-bit byte or 16-bit word from an input port.

SYNTAX

IB <port addr> [] * []

IW <port addr> [] [] * []

OPERATION

To use either the Input Byte or Input Word command, press the corresponding hexadecimal key when prompted for command entry. When either the IB or IW key is pressed, the decimal point on the right of the address field will light to indicate that a port address entry is required. Using the hexadecimal keypad, enter the port address of the port to be read. Note that since I/O addressing is limited to 64K (maximum address FFFFH), no segment value is permitted with the port address.

After the port address has been entered, press the “,” key. The input byte or word at the addressed port will be displayed in the data field. Again pressing the “,” key updates the data field display with the current data byte or word at the addressed input port. Pressing the “.” key terminates the command and prompts for command entry.

The SDK-86 includes two 8255A parallel I/O port circuits which can be used with the Input Byte and Input Word commands to input data from peripheral devices. These two circuits are designated P1 and P2. Each circuit, in turn, consists of three individual 8-bit ports which are designated port A, port B, and port C. As shown in Figure 3-4, each port operates independently during byte operations. During word operations, a pair of ports (e.g., P1A and P2A) operate together to form the 16-bit wide data word with P2 corresponding to the low-order byte.

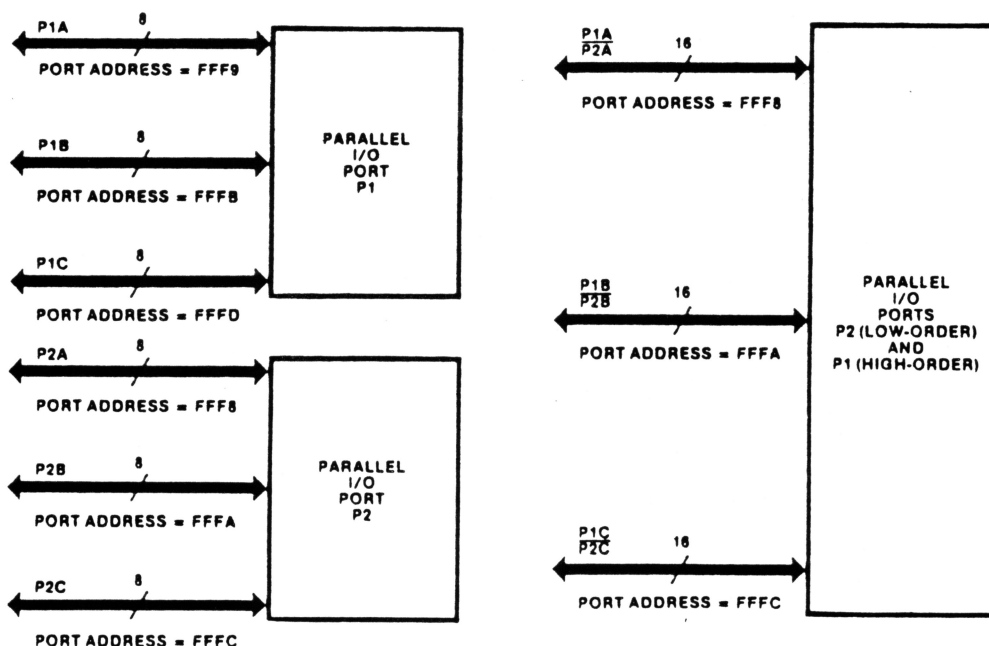


Figure 3-4. Parallel I/O Port Configuration

Table 3-7 defines the individual port addresses. Note that during word operations, the low-order (P2) port address is entered (the corresponding high-order port is addressed automatically).

Table 3-7. Parallel I/O Port Addressing

Port	Address
P2A	FFF8
P1A	FFF9
P2B	FFFA
P1B	FFFB
P2C	FFFC
P1C	FFFD

The parallel I/O port circuits are programmed for input on power-up or whenever the **SYSTM RESET** key is pressed. If the circuit(s) previously has been programmed for output, press the **SYSTM RESET** key (before pressing the command key) or, referring to the next section, output the appropriate byte or word value to the circuit's control port to program the port(s) for input. Additional information regarding the operation of the 8255A parallel I/O port circuit can be found in the *Intel Component Data Catalog*.

EXAMPLES

Example 1: Single Byte Input from Port 0FFH*

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
	-		System Reset
			Input Byte Command
			Port Address
			Input Data Byte
	-		Command Termination/Prompt

* Port 0FFH is not provided on the SDK-86

Example 2: Multiple Word Input from Parallel I/O Ports 1B and 2B

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTM RESET	- 8 6	1 1	System Reset (Initializes Ports For Input)
8 IW/CS			Input Word Command
F			Port B Address
F			
F			
A	F F F A		
,	F F F A	X X X X	Input Data Word From Port
,	F F F A	X X X X	Input Data Word Again
,	F F F A	X X X X	Input Data Word Again
.	-		Command Termination/Prompt

3-9. Output Byte and Output Word Commands

FUNCTION

The Output Byte (OB) and Output Word (OW) commands are used to output a byte or word to an output port.

SYNTAX

OB <port addr> <data> [<data>]* .

OW <port addr> <data> [<data>]* .

OPERATION

To use either command, press the corresponding hexadecimal key when prompted for command entry. When either the **OB** or **OW** key is pressed, the decimal point on the right of the address field will light to indicate that a port address entry is required. Like the Input Byte and Input Word commands, I/O addresses are limited to 64K, and no segment value is permitted. After the port address is entered, press the “,” key. The decimal point on the right of the data field will light to indicate that the data byte or word to be output now can be entered. Using the keypad, enter

the byte or word to be output. After the data is entered, press the "." key to output the byte or word to the port and to terminate the command or press the "," key if additional data is to be output to the addressed port.

As mentioned in the previous section, the Output Byte and Output Word commands can be used to program the 8255A parallel I/O port circuits for input or output as well as to output data to the individual ports. The I/O port circuits are programmed for input on power-up or system reset and consequently first must be programmed for output (by outputting the appropriate data byte or word to the circuit's control port) before data can be output to the associated ports. Table 3-8 defines the control port addressing and associated data byte or word to be output to the control port. Additional information regarding the operation of the 8255A parallel I/O port circuit can be found in the *Intel Component Data Catalog*.

Table 3-8. Control Port Addressing

Port Number	Port Address	Data Byte or Word	
		Input	Output
P2	FFFEH	9BH	80H
P1	FFFFH	9BH	80H
P2/P1	FFFEH	9B9BH	8080H

EXAMPLES

Example 1: Output DI Register Contents to Output Port 0C5H*

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTEM RESET	- E 6	I I	System Reset
9 OW/OS			Output Word Command
C /IP			Output Port Address
5 OB/BP			
,			Enable Data Entry
REG		r	Register Input
7 EW/DI		X X X X	DI Register Contents
.	-		Data Output, Command Termination/Prompt

* Port 0C5H is not provided on the SDK-86

Example 2: Programming Port P1 for Output

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTM RESET	- 0 0 0	1 1	System Reset (Initializes Ports for Input)
5 OB/BP			Output Byte Command
F			P1 Control Port Address
F			
F			
F			
,			Enable Data Entry
8 IW/CS			Control Byte for Output
0 EB/AX			
.	-		Control Byte Output, Command Termination/Prompt

3-10. Go Command

FUNCTION

The Go (GO) command is used to transfer control of the 8086 from the keypad monitor program to a user's program in memory.

SYNTAX

GO [<addr>][<breakpoint addr>]

OPERATION

To use the Go command, press the **GO** key when prompted for command entry. When the key is pressed, the current IP (instruction pointer) register contents are displayed in the address field, the byte contents of the memory location addressed by the IP register are displayed in the data field, and the decimal point at the right of the address field lights to indicate that an alternate start address entry can be entered. If an alternate starting address is required, enter the address from the keypad. (When an address is entered, the data field is blanked.) To begin program execution (at the current instruction or alternate program address), press the "." key. When the key is pressed, the monitor displays an "E" in the most-significant digit of the address field before transferring control to the program.

To illustrate the operation of the Go command, the following sample program can be entered into memory using the Examine Byte command. The program simulates a dice game in which you "throw" a single die. After the program has been entered and after control is passed to the program by the Go command, pressing any keyboard key (other than **SYSTM RESET** or **INTR**) starts the die rolling. Again pressing any key stops the die and displays its value (a number between 1 and 6 appears in the leftmost digit of the address field).

Sample Program

Memory Location	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0100	8C	C9	8E	D9	BA	EA	FF	B0	D3	EE	BA	EA	FF	EC	24	0F
0110	74	FB	E8	28	00	BB	00	00	43	80	FB	07	74	F7	8B	FB
0120	8A	4D	47	90	BA	EA	FF	B0	87	EE	BA	E8	FF	8A	C1	EE
0130	BA	EA	FF	EC	24	0F	74	E0	E8	02	00	EB	CD	BA	EA	FF
0140	B0	40	EE	BA	E8	FF	EC	C3	06	5B	4F	66	6D	7D		

The sample program consists of 4E hexadecimal locations and is entered into memory beginning at location 100H. After you have entered all of the program data, press the "." key to terminate the Examine Byte command and to cause the monitor to request a command entry. When prompted, press the **GO** key and enter the program's starting memory address using a segment value of 10 and an offset value of 0 (10:0). Press the "." key and note that the display is blanked. Now pressing any keyboard key will start the die rolling, and again pressing any key will stop the die. Note that since the sample program uses the most-significant digit of the address field, the "E" is overwritten.

NOTE

After the program is entered into memory, it will remain in memory until power is removed and will not be affected by pressing the **SYSTM RESET** key.

To exit from the executing program and return control to the monitor, press either the **SYSTM RESET** or **INTR** key. If the **SYSTM RESET** key is pressed, the monitor is re-entered and the appropriate 8086 registers are initialized. If the **INTR** key is pressed, the monitor is re-entered, *all* of the 8086 registers are saved, and the monitor prompts for a command entry. When the monitor is re-entered by pressing the **INTR** key, pressing the **GO** key causes the current IP register value (the offset address of the next program instruction to be executed when the **INTR** key was pressed) and the byte contents of that location (addressed by both the IP and CS registers) to be displayed. Pressing the "." key transfers control from the monitor back to the program at the instruction addressed and program execution continues.

The Go command optionally permits a "breakpoint address" to be entered. A breakpoint address has the same affect as pressing the **INTR** key while a program is being executed. To enter a breakpoint address, press the "," key after entering the starting address and enter the breakpoint address. Note that when specifying a breakpoint address, the default segment value is either the starting address segment value (if specified) or the current CS register contents (if a segment value is not specified with the starting address). In addition, the location specified by the breakpoint address must contain the first (opcode or prefix) byte of the instruction. When the "." key is pressed, the monitor replaces the instruction at the breakpoint address with an interrupt instruction and saves the "breakpointed" instruction

before transferring control to the user's program. When the program reaches the breakpoint address, control is returned to the monitor, the breakpointed instruction is restored in the program, all registers are saved, and the monitor prompts for command entry to allow any of the registers to be examined. Note that since the breakpointed instruction is restored when control is returned to the monitor, the breakpoint address must be specified each time the program is to be executed with a breakpoint.

ERROR CONDITIONS

Attempting to breakpoint an instruction in read-only memory.

EXAMPLES

Example 1: Transferring Control to the Sample Program

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
<div>SYSTM RESET</div>	- 0 0	1 1	System Reset
<div>2 GO/CX</div>	0	X X	Go Command (IP register offset address and data contents)
<div>1 ER/BX</div>	1 0		} Segment (CS Register) Address
<div>0 EB/AX</div>	1 0		
<div>:</div>	1 0		Segment/Offset Separator
<div>0 EB/AX</div>	0		Offset Address
<div>.</div>			Control Transferred To 0100H

Example 2: Entering and Executing a Breakpoint in the Sample Program

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTM RESET	- 8 6	1 1	System Reset
2 GO/CX	0.	X X	Go Command
1 ER/BX	1		} Segment (CS Register) Address
0 EB/AX	1 0.		
:	1 0.		Segment/Offset Separator
0 EB/AX	0.		Offset Address
,	.		
2 GO/CX	2.		} Breakpoint Offset Address
0	2 0.		
.			Transfer Control
X	-	b r	Press Any Key. Breakpoint Reached. Command Prompt

3-11. Move Command

FUNCTION

The Move (MV) command permits a block of data to be moved within memory.

SYNTAX

[MV] <start addr> <end addr> <destination addr>

OPERATION

The format of the Move command is unique in that three successive entries are made in the address field. To use the Move command, press the **MV** key when prompted for command entry. When the key is pressed, three decimal points appear in the address field to indicate that three entries are required. Each time an entry is made, the leftmost decimal point goes out so that the number of decimal points lit at any one time indicates the number of entries still required. The entries are, in order:

1. The starting memory address of the block of data to be moved.
2. The ending memory address of the block of data to be moved.

- The starting memory address (destination address) into which the block of data is to be moved.

Note that no segment value is permitted with an ending address and that block moves consequently are limited to 64K bytes.

When the "." key is pressed, the data is moved and the command prompt sign is displayed. Note that when the block of data is moved, the data contained in the original (source) memory locations is *not* altered (unless the destination address falls within the original block of data in which case the overlapping memory locations will be overwritten by the data moved).

Since a move is performed one byte at a time, the Move command can be used to fill a block of memory with a constant. This is accomplished by specifying a destination address that is one greater than the start address. The block of memory locations from start address to end address + 1 will be filled with the value contained in the start address location. (The Examine Byte command can be used to specify the contents of start address.)

ERROR CONDITIONS

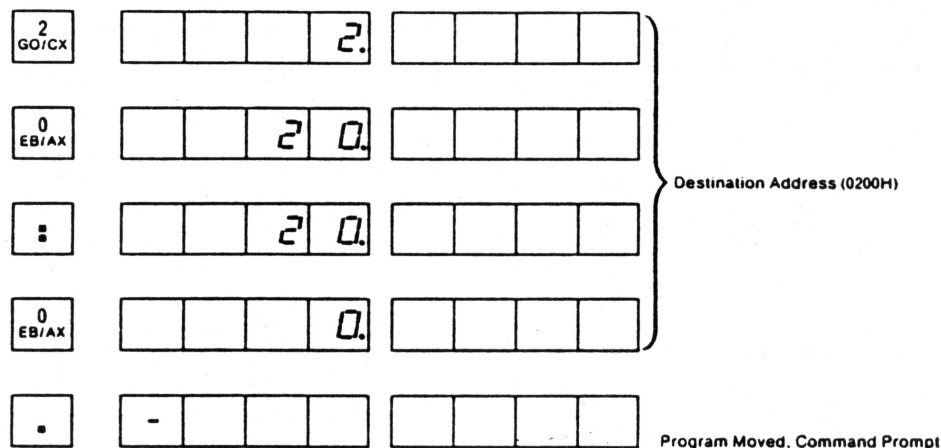
Attempting to move data into read-only or non-existent memory.

EXAMPLES

Example 1: Moving the Dice Program to Location 0200H

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
SYSTM RESET	- 0 0	0 0	System Reset
6 MV/SI	.		Move Command
1 EB/BX	.		Start Address (0100H)
0 EB/AX	0 0		
:	0 0		
0 EB/AX	0 0		
,	.		
4 IB/SP	4		End Offset Address (040H)
D /FL	4 d		
,	.		

Example 1: Moving the Dice Program to Location 0200H (Cont'd.)



3-12. Step Command

FUNCTION

The Step (ST) command permits program instructions in memory to be executed individually. With each instruction executed, control is returned to the monitor from the program.

SYNTAX

ST [**<start addr>**] [**<start addr>**] * □

OPERATION

To use the Step command, press the **ST** key when prompted for command entry. If a starting address other than the address displayed is required, enter the desired address. When the “,” key is pressed, the instruction addressed is executed and the offset address of the next instruction to be executed is displayed in the address field and its associated instruction byte is displayed in the data field. Again pressing the “,” key executes the current instruction and steps the program to the next instruction to be executed.

In the example which follows, the first few instructions of the “rolling dice” program are executed using the step command. The following table represents a listing of the beginning of that program. (A complete listing of the “rolling dice” program is included at the end of this chapter.)

LOCATION	CONTENTS	SYMBOLIC	COMMENTS
00	8CC9	MOV CX,CS	;DEFINE DATA SEGMENT REGISTER EQUAL
02	8ED9	MOV DS,CX	;TO CODE SEGMENT REGISTER
04	BAEAFF	MOV DX,0FFEAH	;CLEAR DISPLAY CONTROL WORD
07	B0D3	MOV AL,0D3H	;TO 8279
09	EE	OUT DX	
READKEY:			
0A	BAEAFF	MOV DX,0FFEAH	;SET UP 8279 COMMAND PORT
0D	EC	IN DX	;READ 8279 FIFO STATUS
0E	240F	AND AL,0FH	;MASK ALL BUT FIFO COUNT
10	74FB	JZ READKEY + 3	;KEEP READING IF ZERO
12	E82800	CALL READATA	;DUMMY READ TO UNLOAD
			;PRESSED KEY FROM FIFO

Note that when the program is stepped from the instruction at 10H, the instruction at 0DH is executed again and the instruction at 12H is not executed. This is caused by the JZ (jump zero) instruction at 10H which, since no key can be pressed to “roll

the dice" (the monitor is in control of the keyboard), jumps back to the instruction at 0DH. Continuing to press the "," key will repeat the three instruction sequence (0DH, 0EH, 10H).

RESTRICTIONS

1. If an interrupt occurs prior to the completion of a single-stepped instruction or if a single-stepped instruction generates an interrupt, when the monitor is re-entered, the CS and IP registers will contain the address of the interrupt service routine. Consequently, a type 3 (breakpoint) interrupt instruction (0CCH or 0CDH) should not be single-stepped since its execution would step into the monitor.
2. An instruction that is part of a sequence of instructions that switches between stack segments (i.e., changes the SS and SP register contents) cannot be single-stepped.
3. A MOV or a POP instruction that modifies a segment register cannot be single-stepped. Control is returned to the monitor after the next instruction (the instruction that immediately follows the MOV or POP instruction) is executed.

EXAMPLES

Example 1: Program Stepping

KEY	ADDRESS FIELD	DATA FIELD	COMMENT
3 ST/OX	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 0.	<input type="text"/> <input type="text"/> X X	Step Command
1 ER/BX	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 1.	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Starting Address of Program
0 EB/AX	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 10.	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
:	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 10.	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
0 EB/AX	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 0.	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
,	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 2.	<input type="text"/> <input type="text"/> B E	Next Instruction
,	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 7.	<input type="text"/> <input type="text"/> b 0	
,	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> 9.	<input type="text"/> <input type="text"/> E E	
,	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> A.	<input type="text"/> <input type="text"/> b A	
,	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> d.	<input type="text"/> <input type="text"/> E C	

Example 1: Program Stepping (Cont'd.)

,				E.			2	4
,			1	0.			7	4
,				d.			E	C

3-13. Program Listing

The following is the complete program listing for the "rolling dice" program used in the previous command examples. For more information regarding program operation and interpretation, refer to *The 8086 Family User's Manual* or the *ASM86 Language Reference Manual*, Order Number 9800640.

LOCATION	CONTENTS	SYMBOLIC	COMMENTS
		ASSUME DS:DICE, CS:DICE	
		DICE SEGMENT AT 100	
00	8CC9	MOV CX,CS	;DEFINE DATA SEGMENT
02	8ED9	MOV DS,CX	;REGISTER EQUAL TO
			;CODE SEGMENT REGISTER
04	BAEAFF	MOV DX,0FFEAH	;CLEAR DISPLAY CONTROL
07	B0D3	MOV AL,0D3H	;WORD TO 8279
09	EE	OUT DX	
		READKEY:	
0A	BAEAFF	MOV DX,0FFEAH	;SET UP 8279 COMMAND PORT
0D	EC	IN DX	;READ 8279 FIFO STATUS
0E	240F	AND AL,0FH	;MASK ALL BUT FIFO COUNT
10	74FB	JZ READKEY + 3	;KEEP READING IF ZERO
12	E82800	CALL READATA	;DUMMY READ TO UNLOAD THE
			;PRESSED KEY FROM FIFO
		ZERO:	
15	BB0000	MOV BX,0000H	;INITIALIZE DICE COUNT TO ZERO
		START:	
18	43	INC BX	;INCREMENT DICE COUNT
19	80FB07	CMP BL,07H	;IF COUNT EQUALS 7, RESET
1C	74F7	JZ ZERO	;COUNT TO 0
1E	8BFB	MOV DI,BX	;PUT COUNT IN DI REGISTER
20	8A4D4790	MOV CL,CDTBL[DI-1]	;PUT 7-SEGMENT DISPLAY
			;CODE IN CL REGISTER,
			;USE DI AS POINTER INTO
			;CODE TABLE
24	BAEAFF	MOV DX,0FFEAH	;OUTPUT "WRITE DISPLAY"
27	B087	MOV AL,087H	;CONTROL WORD TO 8279
29	EE	OUT DX	
2A	BAE8FF	MOV DX,0FFE8H	;OUTPUT THE DICE COUNT
2D	8AC1	MOV AL,CL	;TO 8279 DATA PORT
2F	EE	OUT DX	
30	BAEAFF	MOV DX,0FFEAH	;READ 8279 FIFO STATUS
33	EC	IN DX	
34	240F	AND AL,0FH	;MASK ALL BUT FIFO COUNT
36	74E0	JZ START	;KEEP COUNTING IF NO
			;KEY PRESSED
38	E80200	CALL READATA	;DUMMY READ TO UNLOAD
			;THE PRESSED KEY FROM FIFO
3B	EBCD	JMP READKEY	;GO READ NEXT KEY PRESSED,
			;THEN START COUNT AGAIN

```
3D      BAEAFF      READATA:      MOV      DX,0FFE8H      ;OUTPUT "READ DATA"
40      B040        MOV      AL,040H      ;CONTROL WORD TO 8279
42      EE          OUT      DX
43      BAE8FF      MOV      DX,0FFE8H      ;SET UP AND READ 8279
46      EC          IN       DX            ;DATA PORT, RESULT IN
                                         ;AL REGISTER
47      C3          RET
                                         ;RETURN

48      06          CDTBL      DB      06H      ;7-SEGMENT CODE FOR "1"
49      5B          DB      05BH      ;7-SEGMENT CODE FOR "2"
4A      4F          DB      04FH      ;7-SEGMENT CODE FOR "3"
4B      66          DB      066H      ;7-SEGMENT CODE FOR "4"
4C      6D          DB      06DH      ;7-SEGMENT CODE FOR "5"
4D      7D          DB      07DH      ;7-SEGMENT CODE FOR "6"

DICE    ENDS
END
```


4-1. Introduction

This chapter defines the command set and command formats supported by the serial monitor program. The serial monitor program itself is contained in 4K bytes of ROM and is designed to operate with either a teletypewriter or CRT terminal that has been connected to serial interface EIA connector J7. (Refer to Chapter 6, Peripheral Interfacing, for connector pin assignments, cabling and board configuration.) The serial monitor performs all of the commands available with the keyboard monitor program and additionally provides the capability of reading and punching paper tape.

Both the keypad monitor and serial monitor are located in the EPROMs N5 and N6. On power-up, the keypad monitor is in control. To permit operations using the serial monitor, you must execute a GO command to location FE000H. This will transfer control to the serial monitor.

If the primary mode of operation is to be using the serial monitor, the address of the Inter Segment JUMP could be changed. This can be done by reading the EPROMs N5 and N6 into an EPROM burner buffer, erasing the EPROMs, changing the address of the Inter Segment JUMP to FE000H, and reburning the EPROMs.

The first 256 memory locations (locations 0H-0FFH) are reserved for the monitor and user stack area as shown in Figure 4-1. (First user-available memory location in RAM is 0100H.)

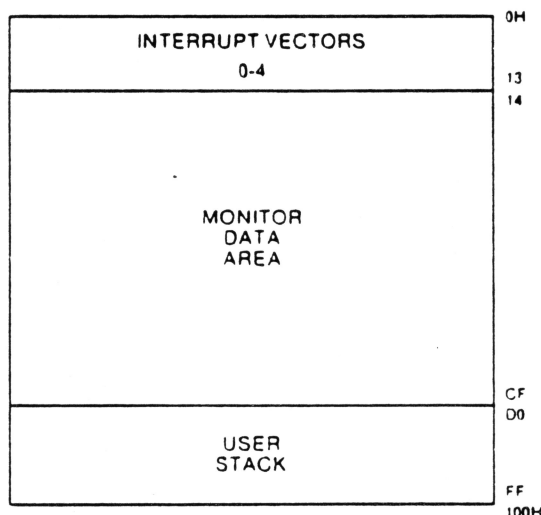


Figure 4-1. Reserved Memory

Whenever the SDK is powered up or whenever the **SYSTEM RESET** key is pressed, the monitor immediately terminates its present activity and jumps to its initialization routine. This routine initializes interrupt vectors 1 through 3 as follows:

- Interrupt 1: Single Step: Used with the Single Step command
- Interrupt 2: NMI (Non-Maskable Interrupt): Monitors INTR key
- Interrupt 3: Breakpoint: Used with the Go command

The routine also initializes the 8086's CS, DS, SS, IP and FL registers to 0H and the SP register to 0100H (base of the stack).

Whenever the monitor is re-entered as a result of a Single Step, NMI or Breakpoint interrupt, the monitor temporarily stores (pushes) the contents of the 8086 registers onto the user stack and subsequently removes (pops) the register contents from the stack before it prompts for command entry. Since the SP register is initialized to 0100H, the initial stack reserved for the user is 48 bytes (locations D0-0FFH) of which 26 bytes must be reserved for the register contents should one of the above interrupts occur.

Following power-on/reset (serial monitor at starting location FF000H) or execution of the Go command (serial monitor at starting location FE000H), the serial monitor displays its sign-on message (SDK-86 MONITOR, Vx.x) on one line and a period (".") on the next line to indicate that it is ready to accept command entry. The monitor's current version number appears in the two least-significant digits of the data field display, the numerals "86" appear in the two least-significant digits of the address field display and, with the exception of the **SYSTM RESET** and **INTR** keys, the keyboard is disabled.

4-2. Monitor Command Structure

When the monitor is ready for a command entry, it outputs a period (".") at the beginning of a new line. This line is referred to as the "command line" and consists of either a one- or two-character command mnemonic followed by from one to three command parameters or "arguments." (If desired for visual separation, a space can be entered between the command mnemonic and the first argument.) When more than one argument is required, a single comma (",") is used between arguments as a delimiter. A command line is terminated either by a carriage return or a comma, depending on the command itself. Commands are executed one at a time and only one command is permitted within a command line.

With the exception of the register abbreviations associated with the X (Examine/Modify Register) command, all arguments are entered as hexadecimal numbers. The valid range of hexadecimal values is from 00 to FF for byte entries and from 0000 to FFFF for word entries (leading zeroes may be omitted). If more than two (byte entries) or four (word entries) digits are entered, only the last two or four digits entered are valid. Address arguments consist of a segment value and an offset value. If a segment value is not entered, the default segment value is the current contents of the code segment (CS) register unless specified otherwise in the command description. When both a segment value and an offset value are entered as an address argument, the first entry is the segment value, and the second entry is the offset value. A colon (":") is entered between the entries as a separator.

Since command execution occurs only after a command terminator is entered, a command entry can be cancelled any time before the terminator is entered by pressing any character that is not legal for the entry expected. When a command is cancelled, the number symbol ("##") is output on the command line, a carriage return and a line feed are issued, and the command prompt character (".") is output on the new line.

4-3. Monitor Commands

The serial monitor is capable of executing ten individual commands. Each command is summarized in Table 4-1 and is described, in detail, within the sections which follow. In both the table and the individual command descriptions, the following command syntax is used:

- [A] Indicates that "A" is optional
- [A]* Indicates one or more optional occurrences of "A"
- Indicates that "B" is a variable
- <cr> Indicates a carriage return is entered

Note that the preceding symbols are used only to clarify the command formats and that they are neither entered nor output on the console device. Also note that in the sections describing the individual commands, output from the monitor is underlined in the examples of command usage, while operator input from the console device is *not* underlined.

Table 4-1. Monitor Command Summary

COMMAND	FUNCTION/SYNTAX
S (Substitute Memory)	Displays/modifies memory locations S[W]<addr>,[<new contents>]*<cr>
X (Examine/Modify Register)	Displays/modifies 8086 registers X<reg>[[<new contents>],]*<cr>
D (Display Memory)	Displays block of memory data D[W]<start addr>,<end addr><cr>
M (Move)	Moves block of memory data M<start addr>,<end addr>,<destination addr><cr>
I (Port Input)	Accepts and displays data at input port I[W]<port addr>,[,]*<cr>
O (Port Output)	Outputs data to output port O[W]<port addr>,<data>[,<data>]*<cr>
G (Go)	Transfers 8086 control from monitor to user program G<start addr>[,<breakpoint addr>]<cr>
N (Single Step)	Executes single user program instruction N<start addr>,[<start addr>]*<cr>
R (Read Hex File)	Reads hexadecimal object file from paper tape into memory R<bias number><cr>
W (Write Hex File)	Outputs block of memory data to paper tape punch W[X]<start addr>,<end addr>,<exec addr><cr>

4-4. Substitute Memory Command

FUNCTION

The Substitute Memory (S) command is used to examine the byte (S) or word (SW) contents of selected memory locations. If the contents of the memory location can be modified (e.g., a RAM location), the contents additionally can be updated with a new data value entered from the console device.

SYNTAX

S[W]<addr>,[<new contents>]*<cr>

OPERATION

To use the Substitute Memory command, enter S or SW when prompted for command entry and enter the address of the memory location to be examined. Note that if a segment address value is not specified, the current contents of the code segment (CS) register are used by default. After the address is entered, enter a comma. The monitor will then output the current data contents of the addressed memory location followed by a dash (the monitor's data entry prompt character) and a space to indicate that the addressed location is open for update. Note that when using the SW command, the byte contents of the next consecutive memory location (memory address + 1) are output first, followed by the byte contents of the actual location addressed. Similarly, when updating memory contents using the SW command, the first byte entry (first two digits) will be written into the next consecutive memory location, and the second byte entry (next two digits) will be written into the addressed memory location.

If only the one memory location is to be examined, enter a carriage return to terminate the command. (If new data was entered, it is updated when the carriage return is entered.) If a series of contiguous memory locations are to be examined and/or updated, enter a comma to advance to the next consecutive memory location (S command) or next two consecutive memory locations (SW command). Again, if the data contents are not to be updated, enter a comma to examine the next memory location, or enter the new data followed by a comma to update the current location and to examine the next location. Entering a carriage return terminates the command.

ERROR CONDITIONS

Attempting to modify a non-existent or read-only (e.g., ROM or PROM) memory location.

EXAMPLES

Example 1: Examine ROM location 0FF000H

```
._S FF00:0, 90- <cr>
```

```
._
```

Example 2: Examine RAM location 050H, relative to the DS register, and update the contents of location 051H to 0F7H

```
._S DS:50, E4- ,  
0051 A4- F7<cr>
```

```
._
```

Example 3: Examine and modify top element on stack

```
._SW SS:SP, C98C- C98A<cr>
```

```
._
```

Example 4: Examine RAM locations 0100H to 0107H, relative to the CS register, and modify location 0105H from 0FBH to 0BBH

```
._SW 100, 3FF3- ,  
0102 FD7B- ,  
0104 FB98- BB98,  
0106 BFF1- <cr>
```

```
._
```

4-5. Examine/Modify Register Command

FUNCTION

The Examine/Modify Register (X) command is used to examine and, if desired, to modify any of the 8086's individual registers or to examine the contents of all of the 8086's registers.

SYNTAX

X[<reg>][[<new contents>],]*<cr>

OPERATION

To use the Examine/Modify Register command, enter X when prompted for command entry. If you only wish to examine the current contents of the registers, enter a carriage return. (The contents of all fourteen registers will be output.) If you wish to examine and optionally to modify the contents of an individual register, enter the register's abbreviation according to Table 4-2.

Table 4-2. Register Abbreviations

Register Name	Abbreviation
Accumulator	AX
Base	BX
Count	CX
Data	DX
Stack Pointer	SP
Base Pointer	BP
Source Index	SI
Destination Index	DI
Code Segment	CS
Data Segment	DS
Stack Segment	SS
Extra Segment	ES
Instruction Pointer	IP
Flag	FL

When a register abbreviation is entered, the monitor outputs an equal sign ("="), the current register contents, the data prompt character ("=") and a space. If you wish to change the register's contents, enter the new contents followed by a comma (to advance to the next "sequential" register) or a carriage return (to terminate the command). The register sequence is the order shown in Table 4-2. Note that the sequence is not circular and that if a comma is entered after the contents of the flag (FL) register are examined or modified, the monitor returns to the command mode. When a carriage return is entered, the register is updated (if new contents were entered) and the monitor returns to the command mode.

EXAMPLES

Example 1: Examine the 8086's registers

```

.X<cr>
AX=89D3 BX=0002 CX= 0010 DX=FFEA SP=0100 BP=D3EB SI=9295
DI=0002 CS=0010 DS=0010 SS=0000 ES=0000 IP=000D FL=F046

```

Example 2: Modify the CS register and examine the next two registers

```

.X CS=0000- 20,
DS=0010- ,
SS=0000- <cr>

```

4-6. Display Memory Command

FUNCTION

The Display Memory (D) command is used to output the contents of a block of memory to the console device.

SYNTAX

D[W]<start addr>[,<end addr>]<cr>

OPERATION

The command provides a line-formatted output of the memory block bounded by *start address* and *end address* (inclusive). Since *end address* is relative to the segment address value specified or implied with *start address* (CS register contents if a segment value is not specified with *start address*), no segment value is permitted with *end address* and block transfers consequently are limited to 64K bytes or 32K words with each command execution.

To use the Display Memory command, enter D (for byte output) or DW (for word output) when prompted for command entry and then enter *start address* of the memory data block. If only one byte or word is to be displayed, enter a carriage return, while if a block of memory is to be displayed, enter *end address* and a carriage return. The monitor will output, beginning on the next line, the starting offset address, the data contents of that location and, when *end address* is specified, the data contents of a number of consecutive memory locations, each separated by a space. The line format output is arranged so that any subsequent lines (if required) will begin with the offset address of the first byte or word in the line and will consist of a maximum of either sixteen byte entries or eight word entries per line.

The Display Memory command can be cancelled or the output can be stopped at any time by entering control characters from the console device. Control-C immediately terminates the command and returns the monitor to the command entry mode. Control-S stops the output, but does not terminate the command. Control-Q resumes output that has been stopped. The only allowed console input following a Control-S is either a Control-Q or Control-C.

ERROR CONDITIONS

End address less than the offset value of *start address*.

EXAMPLES

Example 1: Display contents of locations 09H through 02AH relative to the DS register

```
_D DS:9,2A<cr>
0009 EE BA EA FF EC 24 0F
0010 74 FB E8 27 00 BB 00 00 43 80 FB 07 74 F7 8B FB
0020 8A 4D 46 BA EA FF B0 87 EE BA E8
```

Example 2: Display contents of location 05FFH relative to the CS register

```
_D 5FF<cr>
05FF 08
```

Example 3: Display contents of locations 0FF000H through 0FF02AH in word mode

```
__DW FF00:0,2A<cr>
```

0000	E990	0098	0072	0088	4328	2029	3931	3837
0010	4920	544E	4C45	4320	524F	4050	0000	4000
0020	7F00	007D	8600	5006	0000	4000		

4-7. Move Command

FUNCTION

The Move command is used to move a block of data within memory.

SYNTAX

```
M<start addr>,<end addr>,<destination addr><cr>
```

OPERATION

When using the Move command, the contents of the memory block bounded by *start address* and *end address* (inclusive) are moved to consecutive memory locations beginning at *destination address*. As with the D (Display Memory) command, *end address* is relative to the segment address value specified or implied with *start address* (if no segment value is specified, the CS register contents are used). Consequently, no segment value is permitted with *end address*, and block moves are limited to 64K bytes.

Since a move is performed one byte at a time, the Move command can be used to fill a block of memory with a predefined constant. This is accomplished by specifying a *destination address* which is one greater than *start address*. The block of memory locations from *start address* to *end address* + 1 are filled with the value contained in *start address*. (The S command is used prior to the Move command to define the constant at *start address*.)

ERROR CONDITIONS

Attempting to move data to a read-only (e.g., ROM or PROM) or non-existent memory location.

Specifying an *end address* value which is less than the offset value of *start address*.

EXAMPLES

Example 1: Move the contents of locations 0100H through 014CH to the memory block beginning at 0500H relative to the CS register

```
__M 100,14C,500<cr>
```

Example 2: Move the contents of locations 0200H through 0250H, relative to the DS register, to the memory block starting at the destination address defined by a segment value equal to the ES register plus 010H, and an offset value of 02CH

```
__M DS:200,250,ES + 10:2C<cr>
```

Example 3: Fill memory locations 0300H through 0500H, relative to the CS register, with a constant of 04FH

```

.S 300, 6C- 4F<cr>
.M 300, 4FF, 301<cr>

```

4-8. Port Input Command

FUNCTION

The Port Input (I) command is used to display a byte or word at an input port.

SYNTAX

```
I[W]<port addr>[,]*<cr>
```

OPERATION

The Port Input command inputs a byte (I command) or word (IW command) from the port specified by *port address* and displays the byte or word value on the console device. Since I/O addressing is limited to 64K I/O byte addresses, no segment value is permitted with *port address*. After *port address* is entered, a comma is required to cause the byte or word at the input port to be displayed at the console. Each subsequent comma entered causes the current data at the addressed input port to be displayed on a new line. A carriage return terminates the command and causes the monitor to prompt for command entry.

When using the Port Input command to input data from the 8255A parallel I/O port circuits, refer to Table 4-3 for the assigned addresses of the individual ports. Note that these circuits are programmed for input on power-on/reset and that if they were last programmed for output, they must be reprogrammed for input (see Section 4-9, *Port Output Command*).

Table 4-3. Parallel I/O Port Addressing

Port	Address
P2A	FFF8
P1A	FFF9
P2B	FFFA
P1B	FFFB
P2C	FFFC
P1C	FFFD

When using word input (IW command), the port P2 (low-order byte) address is entered as *port address*.

EXAMPLES

Example 1: Input single word from parallel I/O ports P1A and P2A

```

.IW FFF8,
.C7C5<cr>

```

Example 2: Input multiple bytes from port 02FAH

```

.I 2FA,
.FF,
.FC,
.00,
.B7,
.3F<cr>

```

4-9. Port Output Command

FUNCTION

The Port Output (O) command is used to output a byte or word to an output port.

SYNTAX

O[W]<port addr>,<data>[,<data>]*<cr>

OPERATION

The Port Output command outputs the byte (O command) or word (OW command) entered as data to the output port specified by *port address*. Like the Port Input command, I/O addressing is limited to 64K I/O byte addresses, and no segment value is permitted with *port address*. After entering *port address*, a comma and the data to be output, a carriage return is entered to cause the data to be output to the port and to terminate the command, or a comma is entered to permit subsequent data output to the addressed port. Data can be output repetitively to the port by entering new data followed by a comma. A carriage return following a data entry outputs the data and terminates the command.

As mentioned in the previous section, the two 8255A parallel I/O port circuits are programmed for input on power-on or system reset. Consequently, to use the Port Output command to output data from the parallel I/O ports, the circuits first must be programmed for output. This is accomplished by using the Port Output command to output a control byte (or word) to the 8255A circuit's control port. Table 4-4 defines the control port addressing and the associated data byte or word to be output to the control port to program the circuits for input or output.

Table 4-4. Control Port Addressing

Port Number	Port Address	Control Byte or Word	
		Input Mode	Output Mode
P2	FFFEH	9BH	80H
P1	FFFFH	9BH	80H
P2/P1	FFFEH	9B9BH	8080H

EXAMPLES

Example 1: Program parallel I/O port P2 for output

```
_O FFFE,80<cr>
```

Example 2: Output multiple words to I/O port 020F0H

```
_OW 20F0,BAEA,  
- 4CFF,  
- B0AE,  
- EE47,  
- F9D3<cr>
```

Example 3: Output single byte to parallel I/O port P1C

```
_O FFFD,3C<cr>
```

4-10. Go Command

FUNCTION

The Go (G) command is used to transfer control of the 8086 from the serial monitor program to a user's program in memory.

SYNTAX

G[<start addr>][,<breakpoint addr>]<cr>

OPERATION

To use the Go command, enter G when prompted for command entry. The current IP (instruction pointer) register contents, the data entry prompt character and the byte contents of the location addressed by the IP register are output. If an alternate starting address is required, enter *start address*. To transfer control from the monitor to the program and to begin program execution, enter a carriage return.

To exit from the executing program and to return control to the monitor, press either the **SYSTEM RESET** or the **INTR** key on the keypad. If the **SYSTEM RESET** key is pressed, control is transferred to the monitor program that starts at location FF000H, and the appropriate 8086 registers are initialized. If the **INTR** key is pressed, the program is *interrupted*, the serial monitor is re-entered, *all* of the 8086 registers are saved, and the following message is output followed by a command prompt.

@aaaa:bbbb

In the above message, "aaaa" is the current CS register value, and "bbbb" is the current contents of the IP register. (The combined CS and IP register value is the address of the next program instruction to be executed when the **INTR** key was pressed.) If a subsequent Go command is entered, the current IP register contents ("bbbb") and the data byte addressed by the CS and IP registers are output. When the carriage return is entered, control is transferred back to the program, and execution resumes at the current instruction addressed.

The Go command optionally permits a "breakpoint address" to be entered. A breakpoint address has the same affect as pressing the **INTR** key while a program is being executed. Note that when specifying *breakpoint address*, the default segment value is either the *start address* segment value (if specified) or the current CS register contents (if a segment value is not specified with *start address*). In addition, the location specified by the breakpoint address must contain the first (opcode or prefix) byte of the instruction. When *breakpoint address* is specified, the monitor replaces the instruction at the addressed location with an interrupt instruction and saves the "breakpointed" instruction. When the program reaches *breakpoint address*, control is returned to the monitor, the breakpointed instruction is replaced in the program, all registers are saved, and the monitor outputs the following message followed by a command prompt to allow any of the registers to be examined:

BR @aaaa:bbbb

In the above message, "aaaa" is the current CS register value, and "bbbb" is the current IP register value. (The combined register value is the address of the breakpointed instruction.) If a subsequent Go command is entered, execution resumes at the replaced breakpointed instruction. Note that since the breakpointed instruction is replaced when control is returned to the monitor, *breakpoint address* must be specified each time a program to be breakpointed is executed.

ERROR CONDITIONS

Attempting to breakpoint an instruction in read-only memory.

EXAMPLES

Example 1: Transfer control to the program at 04C0H, relative to the CS register

```
_G 0000D- EC 4C0<cr>
```

Example 2: Transfer control to the program at 10:0H and break at the instruction in location 10:37H

```
_G 010E- 24 10:0,37<cr>  
BR @0010:0037  
_
```

4-11. Single Step Command

FUNCTION

The Single Step (N) command is used to execute a single user-program instruction. With each instruction executed, control is returned to the monitor to allow evaluation of the instruction executed.

SYNTAX

N[<start addr>],[[<start addr>],]*<cr>

OPERATION

To use the Single Step command, enter N when prompted for command entry. The monitor will output the current instruction pointer (IP) register contents (the offset address of the next instruction to be executed) and the instruction byte pointed to by the IP (and CS) register. If execution of an instruction at another address is desired, enter *start address*. If *start address* includes a segment value, both the CS and IP registers are modified. When the comma is entered, the instruction addressed is executed and control is returned to the monitor. The monitor saves all of the register contents and outputs the address (IP register contents) and instruction byte contents of the next instruction to be executed on the following line. Each time a comma is entered, the addressed instruction is executed and the address and instruction byte contents of the next instruction to be executed are output on a new line.

While using the Single Step command to step through a program, a new *start address* can be entered without repeating the command entry. When the comma is entered, the instruction addressed is executed and the next instruction's address and byte contents are output. A carriage return terminates the command.

RESTRICTIONS

1. If an interrupt occurs prior to the completion of a single-stepped instruction or if a single-stepped instruction generates an interrupt, when the monitor is re-entered, the CS and IP registers will contain the address of the interrupt service routine. Consequently, a type 3 (breakpoint) interrupt instruction (0CCH or 0CDH) should not be single-stepped since its execution would step into the monitor.
2. An instruction that is part of a sequence of instructions that switches between stack segments (i.e., changes the SS and SP register contents) cannot be single-stepped.
3. A MOV or a POP instruction that modifies a segment register cannot be single-stepped. Control is returned to the monitor after the next instruction (the instruction that immediately follows the MOV or POP instruction) is executed.

EXAMPLES

Example 1: Single step a series of instructions beginning at 0100H, relative to the CS register

```
.N 0000- 00 100,  
0102- 8E ,  
0107- B0 ,  
0109- EE ,  
010A- BA <cr>
```

Example 2: Single step two instructions and repeat single stepping the second instruction

```
.N 0018- 03 ,  
001A- 40 ,  
001B- 50 1A,  
001B- 50 <cr>
```

4-12. Read Hex File Command**FUNCTION**

The Read Hex File (R) command allows the monitor to read an 8086 or 8080 hexadecimal object file from a paper tape and to load the data read from the file into memory.

SYNTAX

R[<bias number>]<cr>

OPERATION

To use the Read Hex File command, enter R when prompted for command entry. When the tape is loaded in the reader and ready, enter a carriage return. The data read from the file will be written into memory beginning at each record's load address. If the file is in the 8086 format and includes an execution start address record, the CS and IP registers will be updated with the execution address specified in that record. If the file is in the 8080 format and includes an EOF (end-of-file) record, the IP register is updated with the execution address specified in the EOF record. Note that a segment address value is not used with the 8080 file format; the data read is written into memory locations relative to a segment value of zero and, when an EOF record execution address is specified, the CS register is not changed.

When an optional *bias number* is specified, it is added to each record's load address to offset the file in memory.

ERROR CONDITIONS

Tape checksum error.

Attempting to load data into non-existent or read-only memory.

EXAMPLES

Example 1: Read a file and load the data into memory 256 (decimal) bytes above the load addresses specified in the file.

```
.R 100<cr>
```

4-13. Write Hex File Command

FUNCTION

The Write Hex File (W) command allows a block of memory to be output, in either 8086 or 8080 hexadecimal object file format, to a paper tape punch.

SYNTAX

W[X]<start addr>,<end addr>[,<exec addr>]<cr>

OPERATION

To use the Write Hex File command, enter W for 8086 file format or WX for 8080 file format and enter *start address* and *end address* of the memory block to be output. Note that no segment address value is permitted with *end address* (the *start address* segment value is implied) and that if no segment address value is specified with *start address*, the current CS register value is used. When the carriage return is entered, the following information is punched on the paper tape:

- Six inches of leader (60 null characters)
- An extended address record (8086 format only)
- The data contents of the memory block bounded by *start address* and *end address* (inclusive)
- An end-of-file (EOF) record
- Six inches of trailer (60 null characters)

Optionally, an *execution address* can be specified prior to entering the carriage return. This is the memory address that is loaded into the CS and IP registers (IP register only with 8080 format) when the tape is read with the R command. Depending on the format selected, when *execution address* is specified, either an execution start address record containing *execution address* is punched immediately following the tape leader (8086 format) or the offset address value of *execution address* is punched in the EOF record (8080 format).

When using the 8086 format (W command), the *start address* segment value (CS register value if a segment value is not specified) is entered (punched) in the extended address record, and the *start address* offset value is entered in the load address field of the first data record. The segment and offset address values of *execution address* are entered in the execution start address record (CS register contents if a segment address value is not specified with *execution address*).

When using the 8080 format (WX command), the *start address* offset value is punched in the load address field of the first data record. *Execution address*, if specified, is punched in the EOF record. Note that a segment address value is not permitted with *execution address* or *end address* and that the *start address* segment value is used only to define the starting address of the memory block and that it is not punched on the tape.

The Write Hex File command can be cancelled or stopped at any time by entering control characters from the console device. Control-C cancels the command and prompts for new command entry. Control-S stops the output, but does not cancel the command. Control-Q resumes output that has been stopped. The only console input allowed following a Control-S is either a Control-Q or a Control-C.

Additional information regarding Intel object file formats is available in the *MCS 80/85 Absolute Object File Formats Technical Specification*, Order Number 9800183 and the *MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users*, Order Number 9800639.

ERROR CONDITIONS

Specifying a value for *end address* that is less than the offset value of *start address*.

EXAMPLES

Example 1: Output the memory block bounded by 04H and 06DDH, relative to the current CS register, to an 8086 file with an execution address of CS:040H

```
_.W 4,6DD,40<cr>
```

```
._
```

Example 2: Output the memory block bounded by FF200H and FF2FFH to an 8080 file with a starting load address of 0100H and an execution address of 011AH

```
_.WX FF10:100,1FF,11A<cr>
```

```
._
```



5-1. Introduction

When an SDK-86 is interfaced to an Intel Intellec microcomputer development system, the serial loader program that is supplied with the SDK-C86 kit adds two commands to the serial monitor's command set. These commands allow a user to load an ISIS-II file into SDK memory and to transfer the contents of SDK memory to an ISIS-II file.

With the exception of the R and W commands, all of the serial monitor's commands described in Chapter 4 are available with the serial loader program. Except for the operation of the serial loader's load and transfer commands, serial monitor operation with the development system is identical to its operation with a teletypewriter or CRT terminal; characters entered at the system console are input to the SDK and characters output from the SDK are displayed on the system console. The serial loader has the capability of detecting when the serial monitor is prompting for command entry in order to "intercept" a load or transfer command entered from the system console.

When the serial loader intercepts a load command, it opens the specified file for reading and then issues an R (read hex file) command to the serial monitor followed by the data read from the file. When the serial loader intercepts a transfer command, it opens the specified file for writing and issues a W (write hex file) command and the associated command arguments to the serial monitor. The serial monitor responds by outputting the requested data to the serial port, and the serial loader writes the data into the opened file.

5-2. System Operation

Unlike the two monitor programs that reside in ROM, the serial loader program resides on a diskette. Two diskettes are supplied with the SDK-C86 kit. (The contents of both diskettes are identical.) The diskette labeled S.D.D.A. ISIS-II SDK-86 LOADER (part number 9500044) is for single-density drives, and the diskette labeled D.D.D.A. ISIS-II SDK-86 LOADER (part number 9700040) is for double-density drives. Insert the appropriate diskette into the secondary drive and, using the ISIS-II load-and-go command, enter:

```
:Fn:SDK86
```

where n is the drive unit number of the secondary drive. The serial loader will sign-on with the message:

```
ISIS-II SDK-86 LOADER, Vx.x
```

After the serial loader has been loaded (after the sign-on message is displayed) either press the SDK's **SYSTEM RESET** key if the serial monitor is located at FF000H or, if the serial monitor is located at FE000H, execute a Go command from the keypad monitor to transfer control to the serial monitor. The serial monitor's sign-on message (SDK-86 MONITOR, Vx.x) will be displayed on the system console followed by the command prompt character (".") to indicate that the system is ready to accept command entry.

5-3. Loader Commands

As previously mentioned, the serial loader adds two commands to the serial monitor. These commands (Load Hex File and Transfer Hex File) follow the same syntax conventions described in Chapter 4. Additionally, an Exit command is included with the serial loader to return control to ISIS. Table 5-1 defines the serial loader commands and syntax.

Table 5-1. Serial Loader Command Summary

Command	Function/Syntax
T (Transfer Hex File)	Transfers memory block from SDK to ISIS-II file T[X]<start addr>,<end addr>,<filename>[,<exec addr>]<cr>
L (Load Hex File)	Loads ISIS-II file into SDK memory L<filename>[,<bias-number>]<cr>
E (Exit)	Returns control to ISIS E<cr>

5-4. Transfer Hex File Command

FUNCTION

The Transfer Hex File (T) command is used to transfer the contents of a block of SDK memory to an ISIS-II hexadecimal object file in the microcomputer development system.

SYNTAX

T[X]<start addr>,<end addr>,<filename>[,<exec addr>]<cr>

OPERATION

To use the Transfer Hex File command, enter T for 8086 file format or TX for 8080 file format and *start address* and *end address* of the memory block to be transferred and the *filename* of the ISIS-II file to receive the memory data. Note that no segment address value is permitted with *end address* (the *start address* segment value is implied) and that if no segment address value is specified with *start address*, the current CS register value is used. When the carriage return is entered, the data contents of the memory block bounded by *start address* and *end address* (inclusive) are transferred to the file specified.

Optionally, an *execution address* can be specified prior to entering the carriage return. This is the memory address that is loaded into the CS and IP registers (IP register only with 8080 format) when the file is loaded with the L command. Depending on the format selected, when an *execution address* is specified, either an execution start address record containing *execution address* is entered in the file (8086 format) or the offset address value of *execution address* is entered in the EOF record (8080 format).

When using the 8086 format (T command), the *start address* segment value (CS register value if a segment value is not specified) is entered in the file's extended address record, and the *start address* offset value is entered in the load address field of the first data record. The segment and offset address values of *execution address* are entered in the execution start address record (CS register contents if a segment address value is not specified with *execution address*).

When using the 8080 format (TX command), the *start address* offset value is entered in the load address field of the first data record. The *execution address*, if specified, is entered in the EOF record. Note that a segment address value is not permitted with *execution address* or *end address* and that the *start address* segment value is used only to define the starting address of the memory block and that it is not entered in the file.

ERROR CONDITIONS

End address specified is less than the offset value of *start address*

ISIS-II type errors (1-99)

Timeout

Checksum error

EXAMPLES

Example 1: Transfer, in the 8086 file format, the SDK memory contents between locations 0200H and 0600H, relative to the current CS register, into the ISIS-II file named :F2:GARYL

```
_.T 200,600,:F2:GARYL<cr>
```

```
._
```

5-5. Load Hex File Command

FUNCTION

The Load Hex File command loads an ISIS-II hexadecimal object file from the microcomputer development system into SDK memory.

SYNTAX

```
L<filename>[,<bias number>]<cr>
```

OPERATION

To use the Load Hex File command enter L and the ISIS-II *filename* of the file to be loaded. The data read from the specified file will be written into memory beginning at each record's load address. If the file is in the 8086 format and includes an execution start address record, the CS and IP registers will be updated with the execution address specified in that record. If the file is in the 8080 format and includes an EOF (end-of-file) record, the IP register is updated with the execution address specified in the EOF record. Note that a segment address value is not used with the 8080 file format; the data read is written into memory locations relative to a segment value of zero and, when an EOF record execution address is specified, the CS register is not changed.

When an optional *bias number* is specified, it is added to each record's load address to offset the file in memory.

ERROR CONDITIONS

Attempting to load data into non-existent or read-only memory

ISIS-II type errors (1-99)

Timeout

Checksum error

EXAMPLES

Example 1: Load the ISIS-II file named F1:GEL.HEX into SDK memory, 0100H locations above the load addresses in the file.

```
.L F1:GEL.HEX, 100<cr>  
.
```

5-6. Exit Command

FUNCTION

The exit command transfers control from the monitor to ISIS.

SYNTAX

E<cr>

5-7. System I/O Routines

The SDK-86 loader diskette also includes two libraries containing I/O routines for the console and teletypewriter paper tape reader and punch. Although the routines in both libraries have the same names and functions, two libraries are necessary to support the two types of subroutine linkages provided by the 8086 architecture. The routines in SDKIOS.LIB are called with *intra*segment subroutine calls (a PL/M-86 module compiled with the "small" control generates this type of call). The routines in SDKIOL.LIB are called with *inter*segment subroutine calls (a PL/M-86 module compiled with either the "medium" or "large" control generates this type of call). The routines in both libraries are written in PL/M-86. The modules in SDKIOS.LIB are compiled with the "small" control, and the modules in SDKIOL.LIB are compiled with the "large" control. The names assigned to the segments, classes and groups are the standard names generated by the PL/M-86 compiler. (See *MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users*, Order Number 9800639.)

When using the serial loader program, the console input and output routines provided in the library should be used for console I/O. (The loader has special requirements which are met by the library routines.) The paper tape routines (RI and PO) only perform I/O when a teletypewriter is connected directly to the SDK-86's serial interface port.

5-8. Console Input Routine

The console input routine (CI) returns an 8-bit character received from the system console to the caller in the AL register. The AX, CX and DX registers and CPU condition codes are affected by this operation.

When a Control-S (13H) or Control-Q (11H) character is read, it is discarded and another character is read. This function is necessary to support the control character feature of the CO routine.

EXAMPLES

Example 1: PL/M-86 CI Call

```
CI: PROCEDURE BYTE EXTERNAL;
  END CI;
```

```
DECLARE CHAR BYTE;
```

```
CHAR = CI AND 7FH; /* INPUT CHARACTER AND STRIP PARITY BIT */
```

Example 2: ASM86 CI Intrasegment Call

```

                ASSUME DS:DATA,SS:STACK,CS:CODE

DATA            SEGMENT PUBLIC 'DATA'
CHAR            DB      ?
DATA            ENDS

STACK           SEGMENT STACK 'STACK'
                DW      15 DUP ?
BASESTACK       LABEL    WORD
STACK           ENDS

CODE            SEGMENT PUBLIC 'CODE'
                EXTRN    CI:NEAR

INIT:
                MOV      AX,STACK          ; INITIALIZE
                MOV      SS,AX             ; SS
                MOV      SP,OFFSET BASESTACK ; INITIALIZE SP
                MOV      AX,DATA           ; INITIALIZE
                MOV      DS,AX             ; DS

                .
                .
                CALL     CI                ; INPUT CHARACTER FROM CONSOLE
                AND      AL,7FH            ; STRIP OFF PARITY BIT
                MOV      CHAR,AL           ; SAVE CHARACTER
                .
                .

CODE            ENDS
                END      INIT

```

Example 3: ASM86 CI Intersegment Call

```

        EXTRN    CI:FAR

        ASSUME   DS:MYDATA,SS:STACK,CS:MYCODE

MYDATA  SEGMENT 'DATA'
CHAR    DB      ?
MYDATA  ENDS

STACK   SEGMENT STACK 'STACK'
        DW      16 DUP ?
BASESTACK LABEL WORD
STACK   ENDS

MYCODE  SEGMENT 'CODE'
INIT:
        MOV     AX,STACK           ; INITIALIZE
        MOV     SS,AX             ; SS
        MOV     SP,OFFSET BASESTACK ; INITIALIZE SP
        MOV     AX,MYDATA         ; INITIALIZE
        MOV     DS,AX             ; DS

        CALL    CI                ; INPUT CHARACTER FROM CONSOLE
        AND     AL,7FH            ; STRIP OFF PARITY BIT
        MOV     CHAR,AL           ; SAVE CHARACTER

MYCODE  ENDS
        END     INIT

```

5-9. Console Output Routine

The console output routine (CO) transmits an 8-bit character, passed from the caller on the stack in the low-order byte, to the system console. The AX, CX and DX registers and the CPU conditions codes are affected by this operation.

Before the character is output, the console output routine checks to see if a Control-S has been input. If a Control-S has been input, the monitor waits until a Control-Q is input before outputting the character. Consequently, if a Control-S is entered at the console when console output is pending, the output is stopped temporarily. Entering Control-Q resumes output.

EXAMPLES

Example 1: PL/M-86 CO Call

```

CO:PROCEDURE(X) EXTERNAL;
  DECLARE X BYTE;
  END CO;

DECLARE CHAR BYTE;

CALL CO(CHAR); /* OUTPUT CHARACTER */

```

Example 2: ASM86 CO Intrasegment Call

```

                ASSUME  DS:DATA,SS:STACK,CS:CODE

DATA            SEGMENT PUBLIC 'DATA'
CHAR            DB      ?
DATA            ENDS

STACK           SEGMENT STACK 'STACK'
                DW      15 DUP ?
BASESTACK       LABEL  WORD
STACK           ENDS

CODE            SEGMENT PUBLIC 'CODE'
                EXTRN    CO:NEAR

INIT:
                MOV     AX,STACK           ; INITIALIZE
                MOV     SS,AX             ; SS
                MOV     SP,OFFSET BASESTACK ; INITIALIZE SP
                MOV     AX,DATA           ; INITIALIZE
                MOV     DS,AX             ; DS

                MOV     AL,CHAR           ; LOAD CHARACTER INTO AL
                PUSH    AX                ; PUSH CHARACTER ONTO STACK
                CALL    CO                ; OUTPUT CHARACTER TO CONSOLE

CODE            ENDS
                END      INIT

```

Example 3: ASM86 CO Intersegment Call

```

                EXTRN    CO:FAR
                ASSUME  DS:MYDATA,SS:STACK,CS:MYCODE

MYDATA          SEGMENT 'DATA'
CHAR            DB      ?
MYDATA          ENDS

STACK           SEGMENT STACK 'STACK'
                DW      16 DUP ?
BASESTACK       LABEL  WORD
STACK           ENDS

MYCODE          SEGMENT 'CODE'
INIT:
                MOV     AX,STACK           ; INITIALIZE
                MOV     SS,AX             ; SS
                MOV     SP,OFFSET BASESTACK ; INITIALIZE SP
                MOV     AX,MYDATA         ; INITIALIZE
                MOV     DS,AX             ; DS

                MOV     AL,CHAR           ; LOAD CHARACTER INTO AL
                PUSH    AX                ; PUSH CHARACTER ONTO STACK
                CALL    CO                ; OUTPUT CHARACTER TO CONSOLE

MYCODE          ENDS
                END      INIT

```

5-10. Reader Input Routine

The reader input routine, (RI) returns an 8-bit character received from the paper tape reader to the caller in the AL register. The AX, CX and DX registers and the CPU condition codes are affected by this operation. If a character is not received within two seconds, the carry flag of the FL register is set on return.

EXAMPLES

Example 1: PL/M-86 RI Call

```
RI: PROCEDURE BYTE EXTERNAL;
    END RI;
```

```
DECLARE CHAR BYTE;
```

```
CHAR = RI;    /* READ CHARACTER */
IF CARRY THEN GOTO END$OF$FILE;
CHAR = CHAR AND 7FH;    /* STRIP PARITY BIT */
```

Example 2: ASM86 RI Intrasegment Call

```

                ASSUME  DS:DATA,SS:STACK,CS:CODE

DATA            SEGMENT PUBLIC 'DATA'
CHAR            DB      ?
DATA            ENDS

STACK           SEGMENT STACK 'STACK'
                DW      15 DUP ?
BASESTACK       LABEL   WORD
STACK           ENDS

CODE            SEGMENT PUBLIC 'CODE'
EXTRN           RI:NEAR

INIT:
                MOV     AX,STACK           ; INITIALIZE
                MOV     SS,AX              ; SS
                MOV     SP,OFFSET BASESTACK ; INITIALIZE SP
                MOV     AX,DATA            ; INITIALIZE
                MOV     DS,AX              ; DS

                .
                .
                .
                CALL    RI                ; INPUT CHARACTER FROM READER
                JB      DONE               ; EXIT IF NO MORE
                AND     AL,7FH             ; STRIP OFF PARITY BIT
                MOV     CHAR,AL            ; SAVE CHARACTER
                .
                .
                .

CODE            ENDS
                END      INIT
```

Example 3: ASM86 RI Intersegment Call

```

        EXTRN    RI:FAR
        ASSUME   DS:MYDATA,SS:STACK,CS:MYCODE

MYDATA  SEGMENT 'DATA'
CHAR    DB      ?
MYDATA  ENDS

STACK   SEGMENT STACK 'STACK'
        DW      16 DUP ?
BASESTACK LABEL WORD
STACK   ENDS

MYCODE  SEGMENT 'CODE'
INIT:
        MOV     AX,STACK           ; INITIALIZE
        MOV     SS,AX             ; SS
        MOV     SP,OFFSET BASESTACK ; INITIALIZE SP
        MOV     AX,MYDATA         ; INITIALIZE
        MOV     DS,AX             ; DS

        CALL    RI               ; INPUT CHARACTER FROM READER
        JB      DONE             ; EXIT IF NO MORE
        AND     AL,7FH           ; STRIP OFF PARITY BIT
        MOV     CHAR,AL          ; SAVE CHARACTER

MYCODE  ENDS
        END      INIT

```

5-11. Punch Output Routine

The punch output routine (PO) outputs an 8-bit character, passed from the caller on the stack in the low-order byte, to the paper tape punch. The AX and DX registers and the CPU condition codes are affected by this operation.

EXAMPLES

Example 1: PL/M-86 PO Call

```

PO: PROCEDURE(X) EXTERNAL;
    DECLARE X BYTE;
    END PO;

DECLARE CHAR BYTE;

CALL PO(CHAR); /* PUNCH CHARACTER */

```

Example 2: ASM86 PO Intrasegment Call

```

        ASSUME   DS:DATA,SS:STACK,CS:CODE

DATA    SEGMENT PUBLIC 'DATA'
CHAR    DB      ?
DATA    ENDS

```

```

STACK      SEGMENT STACK 'STACK'
            DW      15 DUP ?
BASESTACK LABEL WORD
STACK      ENDS

CODE       SEGMENT PUBLIC 'CODE'
EXTRN      PO:NEAR

INIT:
            MOV      AX,STACK          ; INITIALIZE
            MOV      SS,AX             ; SS
            MOV      SP,OFFSET BASESTACK ; INITIALIZE SP
            MOV      AX,DATA           ; INITIALIZE
            MOV      DS,AX             ; DS

            MOV      AL,CHAR           ; LOAD CHARACTER INTO AL
            PUSH     AX                ; PUSH CHARACTER ONTO STACK
            CALL     PO                ; OUTPUT CHARACTER TO PUNCH

CODE       ENDS
END        INIT

```

Example 3: ASM86 PO Intersegment Call

```

EXTRN      PO:FAR

ASSUME     DS:MYDATA,SS:STACK,CS:MYCODE

MYDATA     SEGMENT 'DATA'
CHAR       DB      ?
MYDATA     ENDS

STACK      SEGMENT STACK 'STACK'
            DW      16 DUP ?
BASESTACK LABEL WORD
STACK      ENDS

MYCODE     SEGMENT 'CODE'
INIT:
            MOV      AX,STACK          ; INITIALIZE
            MOV      SS,AX             ; SS
            MOV      SP,OFFSET BASESTACK ; INITIALIZE SP
            MOV      AX,MYDATA         ; INITIALIZE
            MOV      DS,AX             ; DS

            MOV      AL,CHAR           ; LOAD CHARACTER INTO AL
            PUSH     AX                ; PUSH CHARACTER ONTO STACK
            CALL     PO                ; OUTPUT CHARACTER TO PUNCH

MYCODE     ENDS
END        INIT

```



CHAPTER 6 PERIPHERAL INTERFACING

6-1. Introduction

This chapter provides the necessary information for connecting a teletypewriter/CRT terminal or an Intel Intellec microcomputer development system to the SDK-86's serial interface port. Additionally, the connector pin assignments for the two parallel I/O ports (P1 and P2) and the bus expansion interface are provided.

6-2. Serial Interface Port

The serial interface port uses a DB-25S female type 25 pin EIA connector (J7). The required mating connector (not supplied) is a DB-25P. The pin assignments for the connector are determined by installing shorting plugs at 2- by 18-pin header W1-W18 (located just below serial interface connector J7). The two rows of header pins are grouped into four sets as noted by the silkscreening on the board (TTY, CRT, MDS-TTY and MDS-CRT). Depending on the device to be interfaced, shorting plugs are inserted over all of the pins in the appropriate set. Table 6-1 defines the connector pin assignments and signal names for each set. Note that to use the serial interface, the SDK-86 requires a -12 volt supply.

Table 6-1. Serial Interface Connector J7 Pin Assignments

PIN	HEADER W1-W18 SHORTING PLUG CONFIGURATION			
	TTY	CRT	MDS-TTY	MDS-CRT
1				
2		RECEIVE DATA		TRANSMIT DATA
3		TRANSMIT DATA		RECEIVE DATA
4				
5				
6				
7	GROUND	GROUND	GROUND	GROUND
8				
9				
10				
11				
12	RECEIVE DATA		TRANSMIT DATA	
13	TRANSMIT DATA		RECEIVE DATA	
14				
15				
16	READER CONTROL			
17				
18				
19				
20				
21	READER CONTROL RETURN			
22				
23				
24	RECEIVE DATA RETURN			
25	TRANSMIT DATA RETURN			

The SDK-86's baud rate must be set to match the baud rate of the device connected to the serial interface port. The SDK-86's baud rate is determined by installing a shorting plug at BAUD RATE SELECT header W19-W26 (located towards the center of the Serial Interface area). The available baud rates are individually silkscreened adjacent to their associated header pins. To select the baud rate, simply install a shorting plug over the corresponding pair of header pins. Note that for 110 baud, two shorting plugs are installed.

6-3. SDK-C86 Interface

The SDK-C86 System Design Kit includes a cable for connecting the SDK-86 to any of the Intel Intellec microcomputer development systems. This cable (Intel part number 4001927) is installed between serial interface connector J7 on the SDK-86 and the appropriate connector on the rear of the development system enclosure. Figure 6-1 shows the interface configurations for the various systems and console devices. Referring to the figure, the shorting plug set to be installed on header W1-W18 is shown within the SDK-86 outline as is the required baud rate.

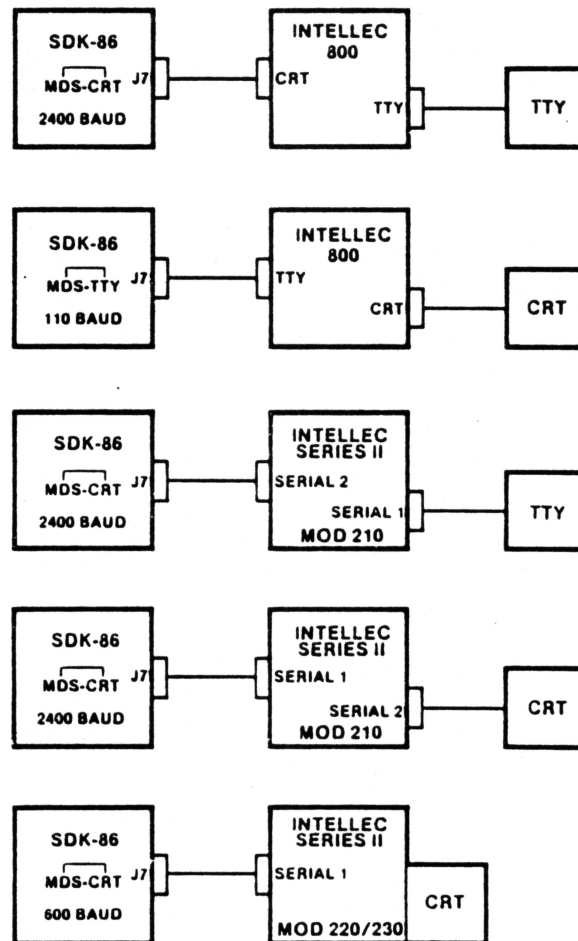


Figure 6-1. Intellec[®] Microcomputer Development System Interfacing

6-4. Parallel I/O Ports

The input/output signals for the two 8255A parallel I/O port circuits are available at the two sets of double row pads to the left of the circuits. As indicated by the silkscreen, the set of pads at J5 corresponds to port circuit P2, and the set of pads at J6 corresponds to port circuit P1. The individual 8-bit ports (A, B and C) also are silkscreened. The pin assignments for the individual port bits are defined in Tables 6-2 (Port 2-J5) and 6-3 (Port 1-J6).

If an off-board peripheral device is to be interfaced to the parallel I/O ports, the 2-by 25-pin headers supplied with the kit can be installed to permit ribbon cable and mass-termination type connectors to be used. Conversely, if the parallel I/O ports are to be interfaced to circuitry installed in the user design area, wire-wrap pins (not supplied) can be installed in the individual pads (from the bottom of the board) to allow wire-wrap connections into the user design area.

Table 6-2. Parallel I/O Port Connector J5 Pin Assignments

Pin	Function	Pin	Function
1	Ground ↑ ↓ Ground	34	Port P2A bit 0
3		38	Port P2A bit 1
5		42	Port P2A bit 2
7		46	Port P2A bit 3
9		48	Port P2A bit 4
11		44	Port P2A bit 5
13		40	Port P2A bit 6
15		36	Port P2A bit 7
17	Ground ↑ ↓ Ground	24	Port P2C bit 0
19		2	Port P2C bit 1
21		4	Port P2C bit 2
23		6	Port P2C bit 3
25		26	Port P2C bit 4
27		28	Port P2C bit 5
29		30	Port P2C bit 6
31		32	Port P2C bit 7
33	Ground ↑ ↓ Ground	10	Port P2B bit 0
35		22	Port P2B bit 1
37		18	Port P2B bit 2
39		14	Port P2B bit 3
41		16	Port P2B bit 4
43		20	Port P2B bit 5
45		12	Port P2B bit 6
47		8	Port P2B bit 7
49	Ground	50	Not Used

Table 6-3. Parallel I/O Port Connector J6 Pin Assignments

Pin	Function	Pin	Function
1	Ground ↑ ↓ Ground	36	Port P1A bit 0
3		40	Port P1A bit 1
5		44	Port P1A bit 2
7		48	Port P1A bit 3
9		50	Port P1A bit 4
11		46	Port P1A bit 5
13		42	Port P1A bit 6
15		38	Port P1A bit 7
17	Ground ↓	26	Port P1C bit 0
19		24	Port P1C bit 1
21		22	Port P1C bit 2

Table 6-3. Parallel I/O Port Connector J6 Pin Assignments

Pin	Function	Pin	Function
23	↑ Ground	20	Port P1C bit 3
25		28	Port P1C bit 4
27		30	Port P1C bit 5
29		32	Port P1C bit 6
31		34	Port P1C bit 7
33	Ground ↑ ↓ Ground	16	Port P1B bit 0
35		12	Port P1B bit 1
37		8	Port P1B bit 2
39		4	Port P1B bit 3
41		6	Port P1B bit 4
43		10	Port P1B bit 5
45		14	Port P1B bit 6
47		18	Port P1B bit 7
49	Ground	2	Not Used

6-5. Bus Expansion Interface

The bus expansion logic, which allows the CPU bus to be extended into the user design area as well as off the board, uses the two sets of double row pads adjacent to the user design area for interfacing. The set of pads designated J1 and J2 provides access to the CPU's data bus and certain control signals while the set of pads designated J3 and J4 provides access to the CPU's 20-bit address bus and the remaining control signals. The signals appearing at J3 and J4 are pin-for-pin identical, while five additional signals are available at J2 that are not available at J1 (the remaining signals are pin-for-pin identical). Table 6-4 lists the pin assignments and corresponding silkscreen signal nomenclature. Note that the signal names in parentheses are the actual signal names appearing on the schematics.

Table 6-4. Bus Expansion Pin Assignments

Pin*	Connector J1/J2 Signal Name	Connector J3/J4 Signal Name
2	D0 (BD0)	$\overline{\text{BHE}}$
4	D1 (BD1)	A0
6	D2 (BD2)	A1
8	D3 (BD3)	A2
10	D4 (BD4)	A3
12	D5 (BD5)	A4
14	D6 (BD6)	A5
16	D7 (BD7)	A6
18	D8 (BD8)	A7
20	D9 (BD9)	A8
22	D10 (BD10)	A9
24	D11 (BD11)	A10
26	D12 (BD12)	A11
28	D13 (BD13)	A12
30	D14 (BD14)	A13
32	D15 (BD15)	A14
34	RST (RESET OUT)	A15
36	PCLK	A16
38	INTR	A17
40	TEST	A18
42	HOLD	A19

*All undesignated pins are logic ground.

Table 6-4. Bus Expansion Pin Assignments (Cont'd.)

Pin*	Connector J1/J2 Signal Name	Connector J3/J4 Signal Name
44	HLDA (BHLDA)	M/ \overline{IO} (BM/ \overline{IO})
46	\overline{DEN} (BDEN/)	\overline{RD} (BRD/)
48	DT/ \overline{R} (BDT/ \overline{R})	\overline{WR} (BWR/)
50	ALE (BALE)	\overline{INTA} (BINTA/)
J2-41	\overline{CSX}	
J2-43	\overline{CSY}	
J2-45	S3 (BS3)	
J2-47	S4 (BS4)	
J2-49	S5 (BS5)	

*All undesignated pins are logic ground.

Since the two sets of signals at J1/J2 and J3/J4 are identical (except for the additional signals at J2), the CPU bus can be routed off-board by installing one of the supplied 2- by 25-pin headers into one of the double row pads in each set (e.g., J2 and J4) and also can be wired into the user design area through the other set (e.g., J1 and J3).

Note that the INTR, TEST and HOLD signals were disabled when the kit was assembled (shorting plugs were installed in W36, W37 and W38) and that if the circuit to be interfaced to the CPU bus requires any of these signals, the corresponding shorting plugs must be removed.



APPENDIX A TELETYPEWRITER MODIFICATIONS

A-1. Introduction

This appendix provides information required to modify a Model ASR-33 Teletypewriter for use with the SDK-86.

A-2. Internal Modifications

WARNING

Hazardous voltages are exposed when the top cover of the teletypewriter is removed. To prevent accidental shock, disconnect the teleprinter power cord before proceeding beyond this point.

Remove the top cover and modify the teletypewriter as follows:

- a. Remove blue lead from 750-ohm tap on current source register; reconnect this lead to 1450-ohm tap. (Refer to figures A-1 and A-2.)
- b. On terminal block, change two wires as follows to create an internal full-duplex loop (refer to figures A-1 and A-3):
 1. Remove brown/yellow lead from terminal 3; reconnect this lead to terminal 5.
 2. Remove white/blue lead from terminal 4; reconnect this lead to terminal 5.
- c. On terminal block, remove violet lead from terminal 8; reconnect this lead to terminal 9. This changes the receiver current level from 60 mA to 20 mA.

A relay circuit card must be fabricated and connected to the paper tape reader driver circuit. The relay circuit card to be fabricated requires a relay, a diode, a thyrector, a small 'vector' board for mounting the components, and suitable hardware for mounting the assembled relay card.

A circuit diagram of the relay circuit card is included in figure A-4; this diagram also includes the part numbers of the relay, diode, and thyrector. (Note that a 470-ohm resistor and a 0.1 μ F capacitor may be substituted for the thyrector.) After the relay circuit card has been assembled, mount it in position as shown in figure A-5. Secure the card to the base plate using two self-tapping screws. Connect the relay circuit to the distributor trip magnet and mode switch as follows:

- a. Refer to figure A-4 and connect a wire (Wire 'A') from relay circuit card to terminal L2 on mode switch. (See figure A-6.)
- b. Disconnect brown wire shown in figure A-7 from plastic connector. Connect this brown wire to terminal l2 on mode switch. (Brown wire will have to be extended.)
- c. Refer to figure A-4 and connect a wire (Wire 'B') from relay circuit board to terminal L1 on mode switch.

A-3. External Connections

Connect a two-wire receive loop, a two-wire send loop, and a two-wire tape reader control loop to the external device as shown in figure A-4. The external connector pin numbers shown in figure A-4 are for interface with an RS232C type, 25-pin connector.

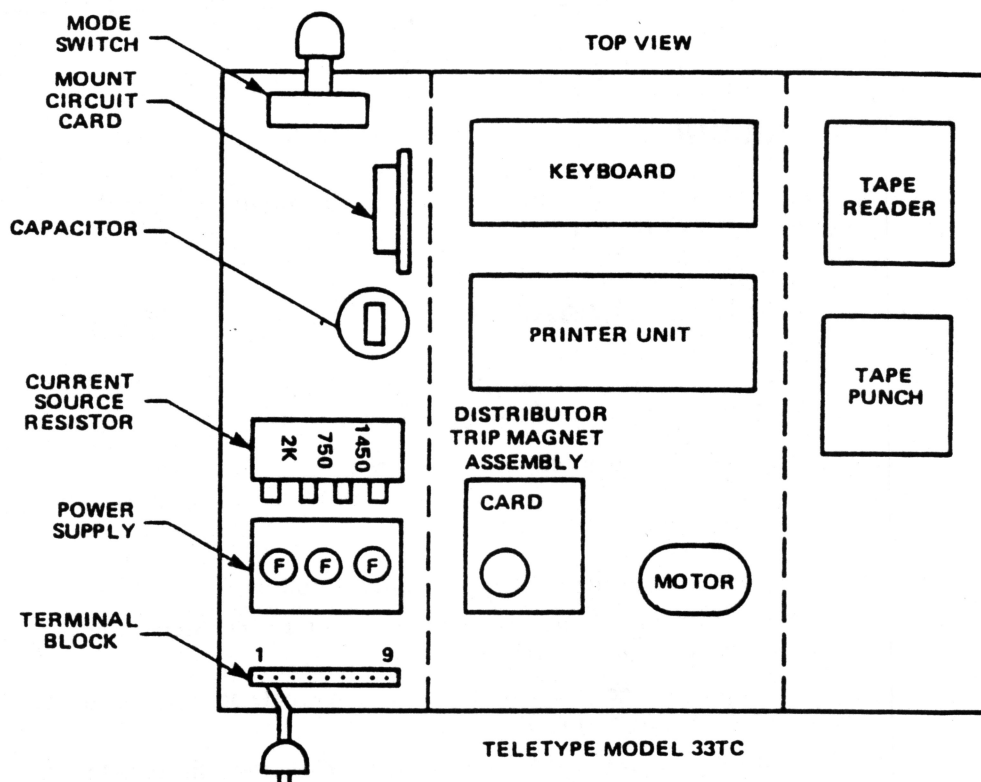


Figure A-1. Teletype Component Layout

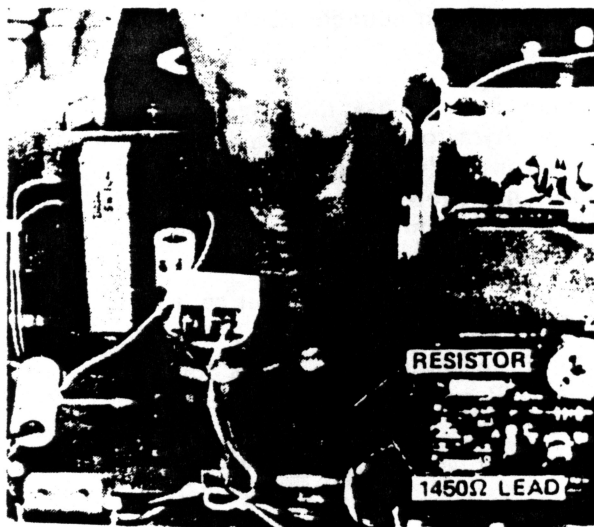


Figure A-2. Current Source Resistor

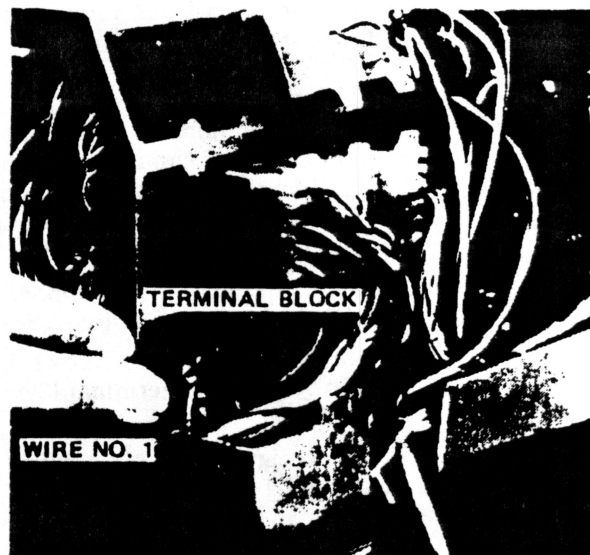


Figure A-3. Terminal Block



B-1. Introduction

This appendix examines the interfacing of additional read-only memory to either the user design area or off-board. The memory devices cited in this appendix are members of the Intel 5 volt ROM and EPROM family and are interfaced to the SDK-86 through the bus expansion logic. For additional information on this ROM/EPROM family, refer to *Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems*, Application Note AP-30.

B-2. ROM/EPROM Family Characteristics

The current and future generations of Intel 5 volt ROMs and EPROMs operate from a single 5 volt source and feature 2-line control (separate Chip Enable and Output Enable inputs) to eliminate bus contention problems in multiple memory systems. While a majority of the currently available devices are contained in 24-pin packages, the 2364 ROM and next generation devices, because of their increased density, require a 28-pin package. Even though the number of pins on the package is increased, compatibility within the entire family is maintained by keeping the functions on the lower 24 pins consistent between the two package sizes as shown in Figure B-1.

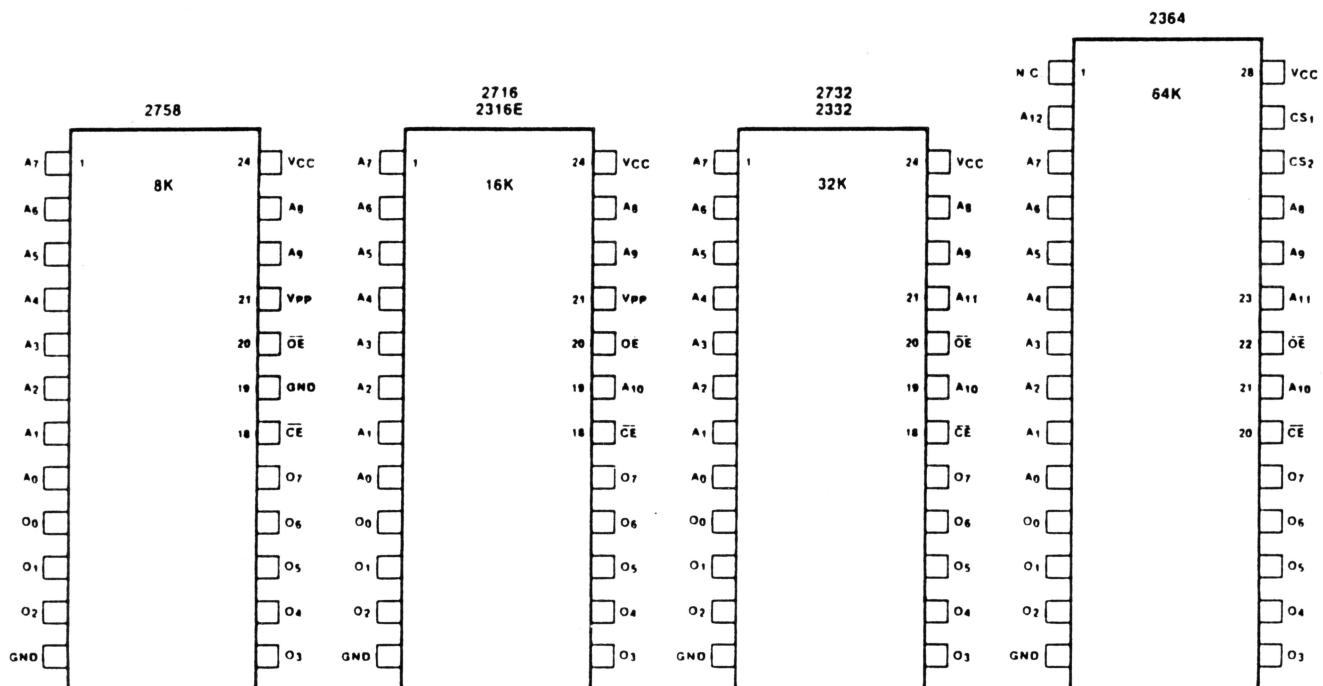


Figure B-1. 5 Volt ROM/EPROM Compatible Family

The following table defines the Intel 5 volt ROM/EPROM family.

Table B-1. Intel™ 5V ROM/EPROM Family

Device	Device Type	Memory Capacity	Memory Organization	Package Size
2758	EPROM	8K	1K × 8	24-pin
2316E	ROM	16K	2K × 8	24-pin
2616	PROM	16K	2K × 8	24-pin
2716	EPROM	16K	2K × 8	24-pin
2332	ROM	32K	4K × 8	24-pin
2732	EPROM	32K	4K × 8	24-pin
2364	ROM	64K	8K × 8	28-pin

B-3. 24-Pin and 28-Pin Sites

As mentioned in the previous section, Intel ROMs and EPROMs, depending on the bit density, are available in either a 24- or 28-pin package. To ensure future upgrading, all designs currently using 24-pin devices should be planned for 28-pin sites. In this way, when a higher-density device of 28 pins is used, the required circuit modifications are limited to only the proper connection of the next high-order address bit(s).

B-4. 8086 Compatibility

Since the 8086 CPU is capable of 16-bit word memory access, ROM and EPROM devices are used in pairs. One device is connected to the low-order data byte (D0-D7), and the other device is connected to the high-order data byte (D8-D15). By offsetting the address bit inputs to the two devices (e.g., A1 address bit connected to the A0 address input of each device) and by using the A0 address bit as a decode input in conjunction with the $\overline{\text{BHE}}$ (Byte High Enable) signal, either the low- or high-order byte or both bytes can be accessed. Table B-2 defines the A0/ $\overline{\text{BHE}}$ truth table used by the SDK-86 for byte and word memory access.

Table B-2. Byte Decoding

Decode Input		Byte Accessed
A0	$\overline{\text{BHE}}$	
0	0	Both Bytes (D0-D15)
1	0	High Byte (D8-D15)
0	1	Low Byte (D0-D7)
1	1	None

B-5. Design Example 1

The following design example (Figure B-2) shows the logic design for the addition of 16, 32 or 64K ROM/EPROM devices in the unassigned SDK-86 memory area immediately below FC000H. In the example, 28-pin sites are used to allow compatibility with a 64K device (2364).

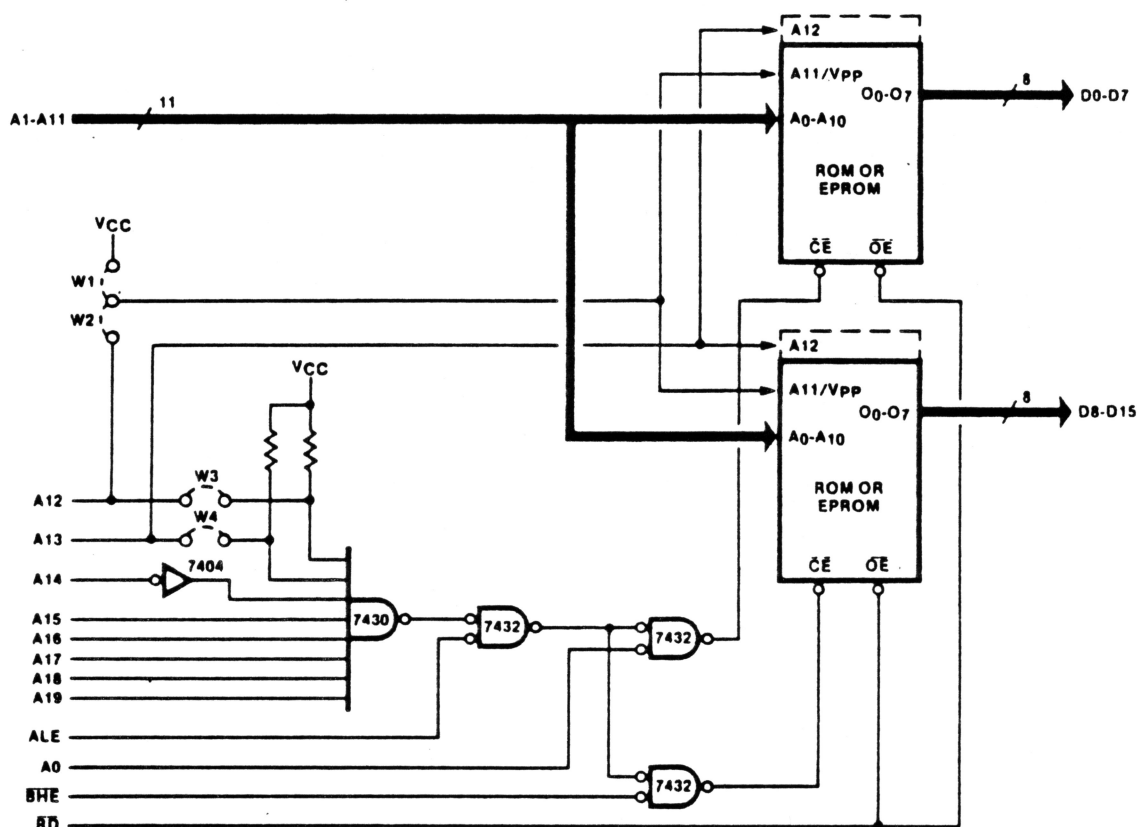


Figure B-2. Design Example 1

In the example, the ALE (Address Latch Enable) signal is gated with the enabled address output from the 7430 for compatibility with edge-enabled devices and is not required for the currently available devices in the family. Table B-3 defines the jumper configurations for 16, 32 and 64K devices. Refer to Table 6-4 for expansion bus interface pin assignments.

Table B-3. Jumper Configurations

Device	Jumpers Installed	Memory Block Selected
2316E/2716	W1, W3, W4	FB000H-FBFFFH
2332/2732	W2, W4	FA000H-FBFFFH
2364	W2	F8000H-FBFFFH

B-6. Design Example 2

The following design example (Figure B-3) shows a logic design which uses the two bus expansion address block decode signals (CSX and CSY) generated by the SDK-86. Since the combined address capability of these two signals is 8K bytes (addresses FC000H-FDFFFH), the maximum memory device that can be used is a 2332/2732, 4K \times 8 ROM/EPROM, and 28-pin sites consequently are not required.

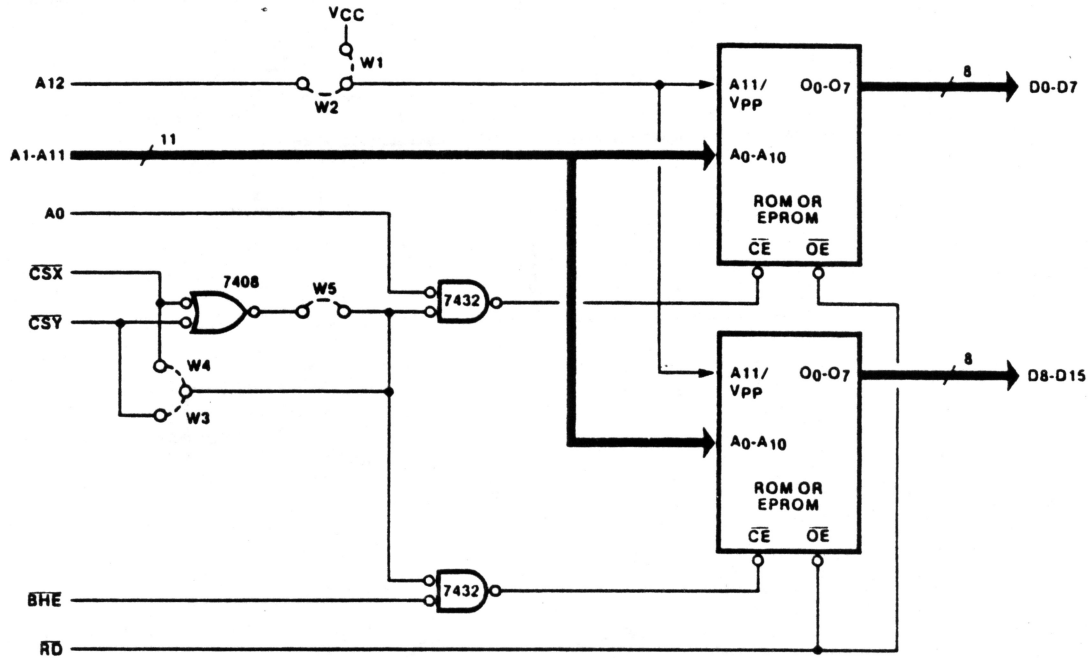


Figure B-3. Design Example 2

Since the existing 16K and 32K devices are static, the ALE signal used in design example 1 is not used. The following table defines the jumper configurations for a 16K or 32K device and the corresponding memory block accessed.

Table B-4. Jumper Configurations

Device	Jumpers Installed	Memory Block Selected
2316E/2716	W1, W4	FD000H-FDFFFH
	W1, W3	FC000H-FCFFFH
2332/2732	W2, W5	FC000H-FDFFFH



UNIVERSITY RESEARCH & DEVELOPMENT ASSOCIATES, INC.
4516 HENRY STREET • SUITE 407 • PITTSBURGH, PENNSYLVANIA 15213 • (412) 683-8732

IMPORTANT NOTICE

If you are going to add an off board memory expansion or an off board peripheral expansion, read this notice. Otherwise, please disregard this notice.

The redesign of the SDK-86 (Revision B) includes a major increase in the size of memory from 4 K Bytes to 16 K Bytes. For many applications this will be sufficient not requiring additional off-board memory. Thus, the Revision B provided an "OFF-BOARD" signal only for use of the SDK-86 to accommodate the existing logic configuration using either the keyboard or the serial monitor.

However, the interfacing aspect of the SDK-86 is important to both teaching and demonstration including memory interfacing. Thus, later versions (Revision C) of the SDK-86 have a fully implemented OFF-BOARD signal.

If you have an SDK-86 supplied with a 74LS00 chip and three (3) 8286 chips in antistatic foam, this is a Revision B model, and the necessary OFF-BOARD circuitry can be added by following the directions and diagram given below as a part of your off board interfacing project.

If your SDK-86 is supplied with the three (3) 8286 chips installed and has a 74LS00 chip (N14) installed, it is a Revision C model, and you can disregard the following discussion.

DIRECTIONS

- (1) Install and solder the three (3) 20 Pin sockets provided at positions A6, A7, and A11.
- (2) Install and solder the 14 pin socket in the 14 pin position below N8. The +5 V and Ground are provided.
- (3) Cut the trace indicated as A in Figure A
- (4) Solder the five (5) wires as shown in Figure A.

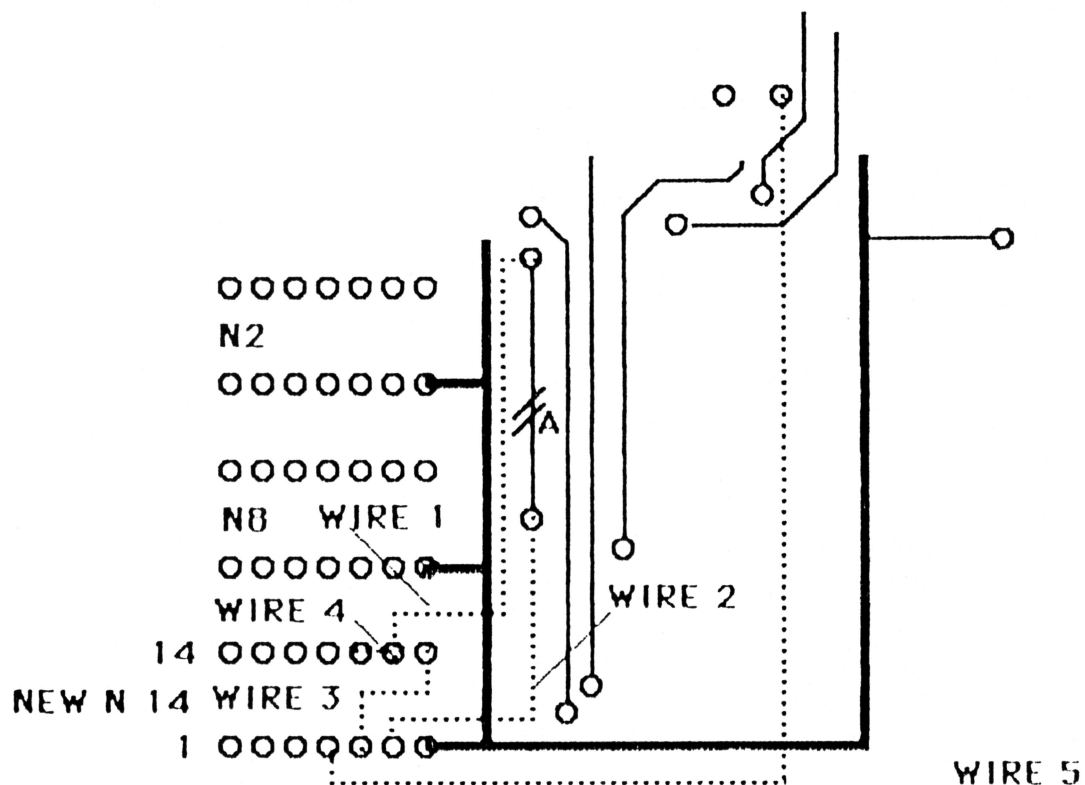


Figure A.

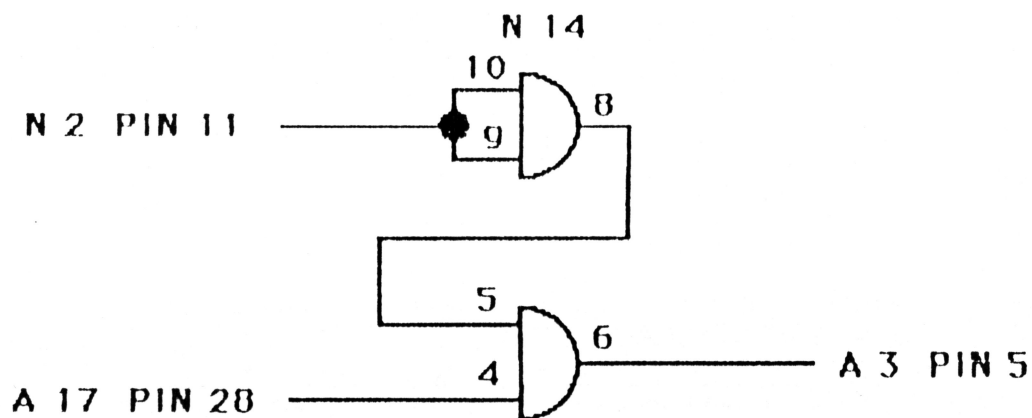


Figure B. Schematic for Figure A PC Modifications